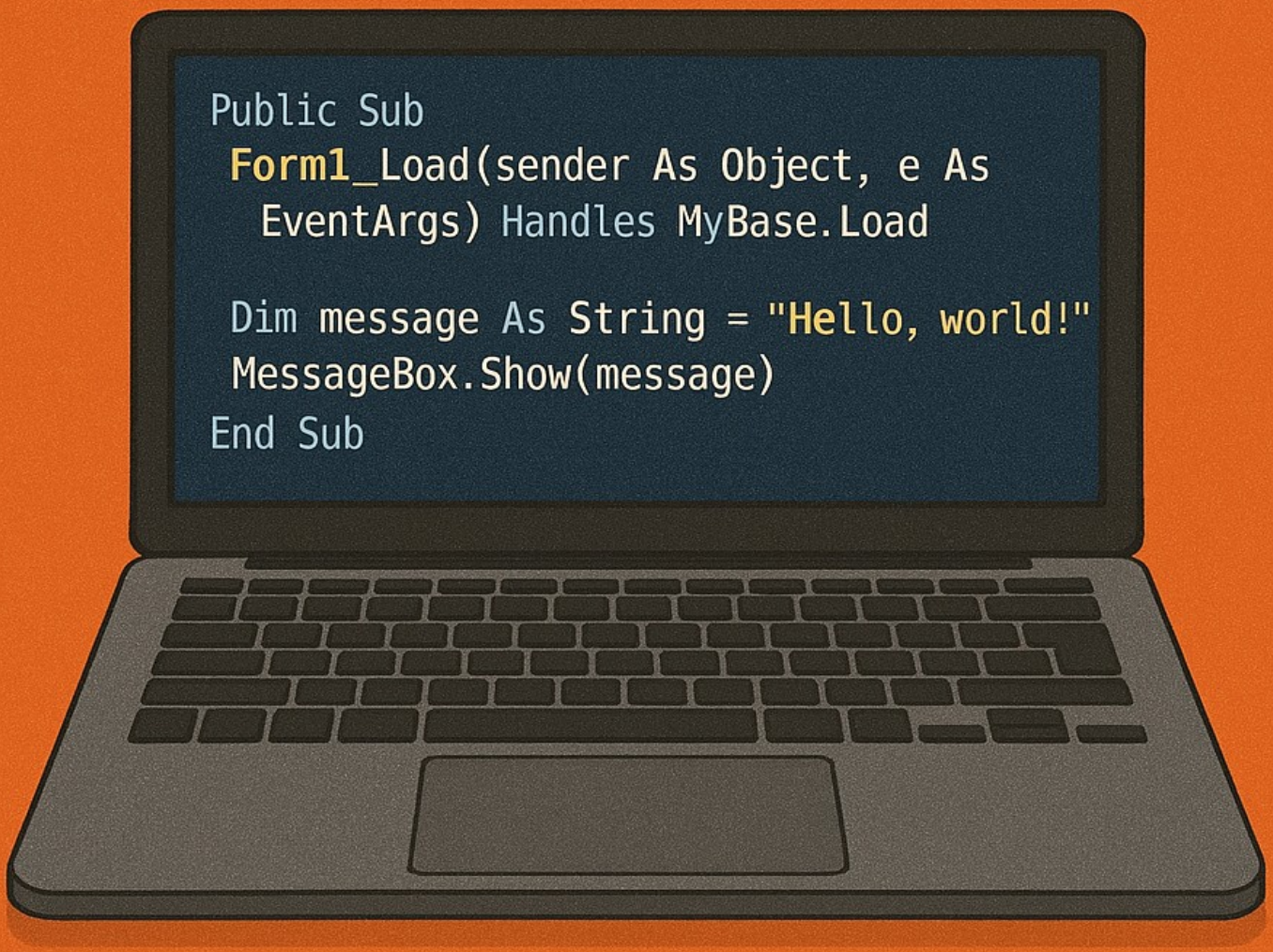
VB.NET

Iniciación



Luis Minguillón Pascual

VB net

Iniciación

Luis Minguillón Pascual

COPYRIGHT
La licencia de este libro electrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos del autor.

Prefacio

Este libro forma parte de una serie que he escrito con el propósito de compartir mis conocimientos sobre programación. No es simplemente una recopilación de códigos o técnicas, sino también un reflejo de un camino personal que comenzó hace muchos años, cuando descubrí el mundo de la programación a través de un lenguaje que muchos recordarán con cariño: QBasic.

QBasic fue el primer lenguaje que aprendí y, sin saberlo entonces, fue también la chispa que encendió mi pasión por la programación. Desde aquel momento, el deseo de comprender cómo funcionan las cosas por dentro y cómo dar vida a las ideas mediante el código se convirtió en una forma de ver el mundo.

A lo largo del tiempo he seguido aprendiendo y trabajando con varios lenguajes: Basic, Visual Basic, Visual Basic .NET, C# y Python. Cada uno me ha aportado herramientas y maneras distintas de pensar, pero todos me han enseñado algo fundamental: que el conocimiento no es una posesión, no es algo que debamos guardar celosamente. Al contrario, el conocimiento debe compartirse, porque solo cuando lo compartimos somos verdaderamente libres. Y es precisamente esa libertad la que intento cultivar en estas páginas.

Este libro no es solo técnico; es una invitación. Una invitación a aprender, a experimentar y a pensar como programador o programadora. Está dirigido tanto a quienes están comenzando como a quienes buscan reforzar sus habilidades en Visual Basic .NET, un lenguaje que sigue siendo potente, claro y muy adecuado para desarrollar aplicaciones de escritorio con interfaz gráfica.

Quiero aprovechar estas líneas también para agradecer profundamente a mi familia, que siempre ha creído en mí, y de manera muy especial a mi mujer, Marisa, por su apoyo incondicional durante todo el proceso de elaboración de este libro. Sin su comprensión, paciencia y aliento constante, este proyecto no habría sido posible.

Gracias por abrir este libro. Ojalá encuentres en él no solo conocimiento, sino también inspiración.

Luis Minguillón Pascual

Índice del libro "VB net Iniciación"

- **Prefacio**
- **1. Fundamentos de Visual Basic .NET**
    1. Introducción
    2. Lenguaje de programación orientado a objetos
    3. Desarrollado por Microsoft, parte del entorno .NET
    4. Permite crear diversas aplicaciones (consola, escritorio, web, servicios)
    5. Diseñado para ser fácil de aprender
    2. Estructura básica de un programa en VB.NET
    3. `Module`, `Sub Main()`, `Console.WriteLine`
    3. Variables y tipos de datos
    4. `Dim`, `Integer`, `String`, `Double`, `Boolean`
    4. Estructuras de control
    5. Condicionales: `If...Then...Else`, `Select Case`
    6. Bucles: `For...Next`, `While...End While`
    5. Funciones y procedimientos (`Function` y `Sub`)
    6. Programación orientada a objetos (`Class`, `Property`, `Sub`)
    7. Manejo de errores (`Try...Catch...Finally`)
    8. Entrada y salida (`Console.Write`, `Console.ReadLine`)
    9. Comentarios en el código
    10. Conclusión
- **2. Primeros Pasos con Visual Basic .NET**
    - ¿Qué es Visual Basic .NET?
        1. Requisitos para comenzar
        2. Visual Studio (IDE)
        3. .NET Framework / .NET Core / .NET 5+ (Plataforma)
        4. Sistema operativo Windows (opcional con .NET Core)
        5. Instalación básica de Visual Studio
        2. Crear tu primer proyecto en VB.NET
        3. Pasos para crear una aplicación de consola
        3. Estructura básica de un programa en VB.NET
        4. `Module Module1`, `Sub Main()`, `Console.WriteLine`, `Console.ReadLine`
        4. Variables y tipos de datos
        5. `String`, `Integer`, `Double`, `Boolean`
        5. Estructuras de control
        6. `If` condicional
        7. `For` bucle
        8. `While` bucle
        6. Crear funciones (`Function`)
        7. Manejo de errores (`Try...Catch...Finally`)

8. Entrada del usuario (`Console.Write`, `Console.ReadLine`)
    - Conclusión
- **3. Tipos de Datos y Variables en Visual Basic .NET**
    1. ¿Qué es una variable?
    2. Declaración de variables (`Dim`)
    2. Reglas para nombrar variables
    3. Tipos de datos en VB.NET (Tabla de tipos comunes: Byte, Short, Integer, Long, Single, Double, Decimal, Char, String, Boolean, Date, Object)
    4. Conversión de tipos (`CInt`, `CStr`, `CDec`)
    5. Inferencia de tipos (con `Option Infer`)
    6. Constantes (`Const`)
    7. Ámbito de las variables (Local, De clase, Global)
    8. Buenas prácticas
- **4. Tipos de Operadores y Expresiones en Visual Basic .NET**
    1. ¿Qué es una Expresión?
    2. Tipos de Operadores en Visual Basic .NET
    3. 2.1 Operadores Aritméticos (+, -, *, /, , Mod, ^)
    4. 2.2 Operadores de Comparación (=, <>, >, <, >=, <=, Is)
    5. 2.3 Operadores Lógicos (And, Or, Not, AndAlso, OrElse, Xor)
    6. 2.4 Operadores de Asignación (=, +=, -=, *=, /=, =, ^=, &=)
    7. 2.5 Operadores de Concatenación (&, +)
    8. 2.6 Operadores de Bits (And, Or, Xor, Not, <<, >>)
    3. Precedencia de Operadores
    4. Expresiones Compuestas
    - Conclusión
- **5. Estructuras de control en Visual Basic .NET**
    1. Introducción
    2. Estructuras condicionales, repetitivas, de selección múltiple
    2. Estructuras Condicionales
    3. 2.1 `If...Then`
    4. 2.2 `If...Then...Else`
    5. 2.3 `If...Then...ElseIf`
    6. 2.4 Operador Ternario (`IIf` o `If Inline`)
    3. Estructuras de Selección Múltiple
    4. 3.1 `Select Case`
    4. Estructuras Repetitivas (Bucles)
    5. 4.1 `For...Next`
    6. 4.2 `For...Each`
    7. 4.3 `While...End While`
    8. 4.4 `Do While / Do Until`
    5. Salida Anticipada de Bucles
    6. 5.1 `Exit For / Exit Do`
    7. 5.2 `Continue For / Continue Do`

6. Conclusión

- **6. Procedimientos y Funciones en Visual Basic .NET**

    1. Introducción
    2. Tipos de procedimientos
    3. 2.1 Subprocedimientos (`Sub`)
    4. 2.2 Procedimientos con parámetros
    3. Funciones (`Function`)
    4. Parámetros por valor y por referencia (`ByVal` y `ByRef`)
    5. Funciones con valores por defecto (`Optional`)
    6. Funciones anidadas (a partir de VB 15.0)
    7. Buenas prácticas
    8. Ejercicio práctico (Calcular área de rectángulo)

- **7. Arreglos y Colecciones en Visual Basic .NET**

    1. ¿Qué es un Arreglo?
    2. Declaración y uso básico
    3. Arreglos multidimensionales
    4. Arreglos dinámicos (`ReDim Preserve`)
    2. ¿Qué es una Colección?
    3. 2.1 `ArrayList`
    4. 2.2 `List(Of T)` (Genérica y tipo seguro)
    5. 2.3 `Dictionary(Of TKey, TValue)`
    3. Comparación entre Arreglos y Colecciones
    4. Ejemplo práctico con `List(Of String)`
    5. Conclusión

- **8. Manejo de cadenas de texto en Visual Basic .NET**

    1. Introducción
    2. Declaración de cadenas
    3. Concatenación de cadenas (&, +)
    4. Longitud de una cadena (`.Length`)
    5. Acceder a caracteres individuales
    6. Subcadenas (`.Substring`)
    7. Conversión a mayúsculas y minúsculas (`.ToUpper()`, `.ToLower()`)
    8. Búsqueda de texto (`.IndexOf()`)
    9. Reemplazo de texto (`.Replace()`)
    10. Eliminación de espacios (`.Trim()`, `.TrimStart()`, `.TrimEnd()`)
    11. Comparación de cadenas (`=`, `String.Compare`)
    12. Dividir una cadena (`.Split()`)
    13. Unir elementos en una cadena (`String.Join()`)
    14. Verificar contenido (`.Contains()`, `.StartsWith()`, `.EndsWith()`)
    15. Formateo de cadenas (`String.Format()`, interpolación de cadenas)
    16. Ejemplo completo
    17. Conclusión

- **9. Clases y Objetos en Visual Basic .NET**

- Introducción
    1. Definición de una Clase (`Class`, `Property`, `Sub`)
    2. Crear un Objeto (Instancia de una Clase) (`New`)
    3. Constructores (`Sub New()`)
    4. Encapsulamiento (Private, Public, Friend)
    5. Herencia (`Inherits`)
    6. Polimorfismo (`Overridable`, `Overrides`)
    7. Clases y Métodos Compartidos (`Shared`)
- Conclusión

- **10. Encapsulamiento, Herencia y Polimorfismo en Visual Basic .NET**
    1. Encapsulamiento (Private, Public, Property Get/Set)
    2. Herencia (`Inherits`, `Overridable`, `Overrides`)
    3. Polimorfismo (Sobrecarga y Sobrescritura)
- Conclusión

- **11. Interfaces y Clases Abstractas en Visual Basic .NET**
    1. Introducción
    2. ¿Qué es una clase abstracta? (`MustInherit`, `MustOverride`)
    3. ¿Qué es una interfaz? (`Interface`, `Implements`)
    4. Diferencias clave (Tabla comparativa)
    5. Cuándo usar una u otra
    6. Ejemplo combinado
    7. Conclusión

- **12. Manejo de Errores y Excepciones en Visual Basic .NET**
    1. Introducción
    2. Tipos de errores (Compilación, Ejecución, Lógicos)
    3. Estructura `Try...Catch...Finally`
    4. Objeto `Exception` (Message, StackTrace, InnerException, Source)
    5. Jerarquía de Excepciones Comunes
    6. Bloque Múltiple `Catch`
    7. Bloque `Finally`
    8. Lanzamiento de Excepciones (`Throw New Exception`, `Throw`)
    9. Excepciones Personalizadas
    10. Manejo de errores con `On Error` (No recomendado)
    11. Buenas prácticas

- **13. Acceso a Archivos en Visual Basic .NET**
    1. Introducción
    2. Espacio de nombres necesario (`Imports System.IO`)
    3. Escribir en un archivo de texto (`File.WriteAllText`, `File.AppendAllText`)
    4. Leer un archivo de texto (`File.ReadAllText`, `File.ReadAllLines`)
    5. Leer y escribir usando `StreamReader` y `StreamWriter`
    6. Verificar existencia y eliminar archivos (`File.Exists`, `File.Delete`)

7. Crear y manejar directorios (`Directory.Exists`, `Directory.CreateDirectory`)
8. Buenas prácticas
9. Ejemplo completo

- **14. Conexión y manejo de bases de datos con Visual Basic .NET**

1. Introducción (ADO.NET)
2. Requisitos previos
3. Componentes principales de ADO.NET (`SqlConnection`, `SqlCommand`, `SqlDataReader`, `SqlDataAdapter`, `DataSet`)
4. Ejemplo de conexión a una base de datos SQL Server
5. Ejemplo: Leer datos de una tabla
6. Insertar datos
7. Actualizar datos
8. Eliminar datos
9. Uso de `DataSet` y `DataAdapter` (modo desconectado)
10. Conclusión

- **15. Creación de Interfaces Gráficas (Windows Forms) en Visual Basic .NET**

1. Introducción a Windows Forms
2. Requisitos Previos
3. Estructura Básica de un Formulario
4. Controles Comunes y su Uso (Label, TextBox, Button, CheckBox, ComboBox, ListBox, PictureBox)
5. Manejo de Eventos (Click, TextChanged, CheckedChanged)
6. Diseño del Formulario
7. Validación de Datos
8. Ventanas Múltiples (`.Show()`)
9. Guardar y Cargar Información (en archivos)
10. Compilar y Ejecutar la Aplicación
11. Ejercicio Práctico (Solicitar nombre y edad)
12. Conclusión

- **16. Aplicaciones Multinivel y Modulares en Visual Basic .NET**

1. Introducción
2. ¿Qué es una Aplicación Multinivel? (UI, BLL, DAL)
3. ¿Qué es una Aplicación Modular?
4. Estructura de una Aplicación Multinivel Modular en VB.NET
5. Ejemplo Básico (Capa de Entidades, DAL, BLL, UI)
6. Buenas Prácticas (Interfaces, SOLID, Inyección de dependencias)
7. Herramientas Sugeridas (Visual Studio, SQL Server, Entity Framework)
8. Conclusión

- **17. Integración de Librerías Externas en Visual Basic .NET**

1. ¿Qué es una librería externa? (DLLs, Terceros, No administradas)
2. Tipos de librerías en VB.NET (DLL, NuGet, COM, No administrada/API de Windows)

3. Cómo agregar una librería .DLL en VB.NET
4. Cómo instalar una librería desde NuGet
5. Usar librerías no administradas (DLLs nativas con `DllImport`)
6. Buenas prácticas
7. Errores comunes
8. Conclusión

- **18. Proyecto de ejemplo paso a paso**
    - Ejercicio 1: Mostrar un mensaje por consola ("Hola, mundo")
    - Ejercicio 2: Sumar dos números ingresados por el usuario
    - Ejercicio 3: Determinar si un número es par o impar
    - **100 Ejercicios Progresivos en Visual Basic .NET**
        1. Ejercicio 1: Mostrar un mensaje por consola
        2. Ejercicio 2: Sumar dos números ingresados
        3. Ejercicio 3: Par o impar
        4. Ejercicio 4: Número mayor
        5. Ejercicio 5: Tabla de multiplicar
        6. Ejercicio 6: Sumar números hasta cero
        7. Ejercicio 7: Contar números positivos
        8. Ejercicio 8: Factorial de un número
        9. Ejercicio 9: Invertir un número
        10. Ejercicio 10: Números primos hasta N
        11. Ejercicio 11: Promedio de números
        12. Ejercicio 12: Número Fibonacci en posición N
        13. Ejercicio 13: Palíndromo (Palabra)
        14. Ejercicio 14: Contar vocales en una cadena
        15. Ejercicio 15: Contar palabras en una frase
        16. Ejercicio 16: Ordenar un arreglo de números
        17. Ejercicio 17: Buscar un número en un arreglo
        18. Ejercicio 18: Sumar elementos pares de un arreglo
        19. Ejercicio 19: Número más pequeño en un arreglo
        20. Ejercicio 20: Conversión de grados Celsius a Fahrenheit
        21. Ejercicio 21: Calcular área de un círculo
        22. Ejercicio 22: Número invertido como cadena
        23. Ejercicio 23: Generar un arreglo con números aleatorios
        24. Ejercicio 24: Sumar elementos de un arreglo
        25. Ejercicio 25: Convertir decimal a binario
        26. Ejercicio 26: Encontrar máximo y mínimo en arreglo
        27. Ejercicio 27: Convertir número a texto (1 a 5)
        28. Ejercicio 28: Comprobar año bisiesto
        29. Ejercicio 29: Suma de dígitos
        30. Ejercicio 30: Crear y mostrar matriz 3x3
        31. Ejercicio 31: Sumar diagonal principal de matriz
        32. Ejercicio 32: Contar caracteres en una cadena
        33. Ejercicio 33: Contar caracteres específicos
        34. Ejercicio 34: Convertir texto a mayúsculas

35. Ejercicio 35: Convertir texto a minúsculas
36. Ejercicio 36: Sumar matriz 3x3
37. Ejercicio 37: Mostrar letras con índice
38. Ejercicio 38: Validar si una cadena es un palíndromo
39. Ejercicio 39: Contar vocales en una cadena
40. Ejercicio 40: Calcular factorial de un número

- **19. Manejo de Fechas en Visual Basic .NET**

    1. Introducción (`Date`, `DateTime`)
    2. Declaración y Asignación de Fechas (`Date.Now`, `#MM/dd/yyyy#`, `New Date()`)
    3. Formatos de Fecha (`ToShortDateString`, `ToLongDateString`, `.ToString("dd-MM-yyyy")`)
    4. Operaciones con Fechas (`AddDays`, `AddMonths`, `AddYears`, `DateDiff`)
    5. Comparación de Fechas
    6. Extraer Componentes de una Fecha (`.Day`, `.Month`, `.Year`, `.Hour`, `.Minute`)
    7. Conversión de Cadenas a Fecha (`Date.Parse`, `Date.TryParse`)
    8. Fechas y Funciones de Calendario (`CultureInfo`, `Calendar`)
    9. Ejemplo Completo
    10. Conclusión

- **20. Manejo del Color en Visual Basic .NET**

    1. Introducción (`System.Drawing.Color`)
    2. Clase `System.Drawing.Color` (`Color.Red`, `Color.FromArgb`, `Color.FromName`)
    3. Usar colores en controles (Form, Label, Button)
    4. Crear colores personalizados con `FromArgb` (con canal alfa)
    5. Paleta de colores predefinidos
    6. Uso en gráficos (GDI+)
    7. Conversión entre colores y cadenas
    8. Usar un `ColorDialog` para elegir colores
    9. Ejemplo completo
    10. Conclusión

- **21. Abrir y Cerrar Formularios en Visual Basic .NET**

    - Introducción (Windows Forms)
        1. Crear y Abrir un Formulario (`.Show()`)
        2. Abrir un Formulario de Modo Modal (`.ShowDialog()`)
        3. Cerrar un Formulario (`Me.Close()`)
        4. Cerrar un Formulario desde Otro Formulario
        5. Ocultar un Formulario en Lugar de Cerrar (`Me.Hide()`)
        6. Ejemplo Completo (Form1 abre Form2 y viceversa)
        7. Consejos Prácticos (`Application.Exit()`, `Using`)

- **22. Uso de Format en Visual Basic .NET**

    - ¿Qué es `Format`?
    - Sintaxis básica
    - Ejemplos comunes (Números, Fechas, Porcentaje)
    - Formatos predefinidos (Tabla: General, Currency, Fixed, Standard, Percent, Short Date, Long Date, Time, yyyy-MM-dd)
    - Consideraciones
- **23. Manejo de Archivos y Directorios en Visual Basic .NET**
    - Introducción (`System.IO`)
    - Clases principales en `System.IO` (Tabla: File, FileInfo, Directory, DirectoryInfo, StreamReader, StreamWriter)
    - Operaciones básicas con archivos (`File.WriteAllText`, `File.ReadAllText`, `File.AppendAllText`, `File.Copy`, `File.Move`, `File.Delete`)
    - Operaciones con directorios (`Directory.CreateDirectory`, `Directory.GetFiles`, `Directory.GetDirectories`, `Directory.Move`, `Directory.Delete`)
    - Lectura y escritura con `StreamReader` y `StreamWriter`
    - Buenas prácticas (`Using`, `Try...Catch`, `File.Exists`, `Directory.Exists`, `Path.Combine`)
    - Ejemplo completo: Crear, escribir y leer archivo
- **24. Controles Comunes de Windows Forms en Visual Basic**
    - Controles Básicos (Tabla: Label, TextBox, Button, CheckBox, RadioButton, PictureBox, GroupBox, Panel, ListBox, ComboBox)
    - Controles Intermedios (Tabla: DateTimePicker, MaskedTextBox, TrackBar, ProgressBar, Timer, TabControl, ListView, TreeView, RichTextBox, ToolTip)
    - Controles Avanzados (Tabla: DataGridView, OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog, MenuStrip, ContextMenuStrip, StatusStrip, NotifyIcon, Chart)
    - Ejemplo Básico de Uso (Formulario con Botón y Etiqueta)
    - Cómo Agregar Controles en Visual Studio
- **25. ¿Qué es un evento?**
    - Ejemplo básico de uso de eventos en VB.NET (Botón con evento Click)
    - Declarar y usar tus propios eventos (`Event`, `RaiseEvent`, `WithEvents`)
    - Palabras clave importantes (Tabla: Event, RaiseEvent, Handles, WithEvents)
    - Eventos comunes en controles de formularios (Tabla: Button Click, TextBox TextChanged, Form Load, Timer Tick)
- **26. Tipos de Métodos en VB.NET**
        1. `Sub` (Subrutina)
        2. `Function`
    - Llamar a un método
    - Métodos con parámetros opcionales (`Optional`)
    - Métodos con parámetros `ByVal` y `ByRef`
    - Métodos en una clase

    - Métodos Estáticos (`Shared`)
- **27. 20 Ejercicios con Métodos en VB.NET**
    - Ejercicio 1: Método Sub simple
    - Ejercicio 2: Método Function con parámetros (Sumar)
    - Ejercicio 3: Método con parámetro opcional (ImprimirMensaje)
    - Ejercicio 4: Método que modifica un parámetro ByRef (Duplicar)
    - Ejercicio 5: Método que devuelve un valor booleano (EsPar)
    - Ejercicio 6: Método sin parámetros que devuelve la fecha actual (FechaActual)
    - Ejercicio 7: Método que imprime los números del 1 al 10 (ImprimirNumeros)
    - Ejercicio 8: Método con varios parámetros (CalcularAreaRectangulo)
    - Ejercicio 9: Método Shared (Matematicas.Cuadrado)
    - Ejercicio 10: Método que concatena cadenas
    - Ejercicio 11: Método para verificar si un string está vacío (EsVacio)
    - Ejercicio 12: Método que recibe un array y devuelve el máximo (Maximo)
    - Ejercicio 13: Método que devuelve la longitud de un string (LongitudCadena)
    - Ejercicio 14: Método que imprime un mensaje N veces (RepetirMensaje)
    - Ejercicio 15: Método que calcula el factorial de un número (Factorial)
    - Ejercicio 16: Método que verifica si un año es bisiesto (EsBisiesto)
    - Ejercicio 17: Método que devuelve la suma de elementos de un array (SumarArray)
    - Ejercicio 18: Método que convierte grados Celsius a Fahrenheit
    - Ejercicio 19: Método con múltiples parámetros opcionales (PresentarPersona)
    - Ejercicio 20: Método que intercambia dos valores usando ByRef (Intercambiar)
- **28. Archivos**
        1. Archivo secuencial con registros (fichas) simples en VB.NET (CSV)
        2. Guardar registros (fichas) en archivo (`File.WriteAllLines`)
        3. Leer fichas desde archivo (`File.ReadAllLines`, `.Split()`)
        2. Usar estructura o clase para la ficha (Clase `Ficha` con propiedades y `ToString()`)
        3. Otras opciones para archivos ficha (Binarios, SQLite, XML, Random Access)
    - Ejemplo: Guardar y cargar una lista de strings en un archivo de texto (`File.WriteAllLines`, `File.ReadAllLines().ToList()`)
- **29. 20 Ejercicios de Archivos en Visual Basic .NET**
    - Ejercicio 1: Crear y escribir texto en un archivo (`StreamWriter`)
    - Ejercicio 2: Leer y mostrar contenido de un archivo (`StreamReader`)
    - Ejercicio 3: Añadir texto a un archivo existente (`StreamWriter` en modo append)
    - Ejercicio 4: Contar líneas de un archivo
    - Ejercicio 5: Buscar una palabra en un archivo (`.Contains()`)
    - Ejercicio 6: Copiar archivo (`File.Copy`)
    - Ejercicio 7: Eliminar un archivo (`File.Exists`, `File.Delete`)
    - Ejercicio 8: Leer archivo completo en una sola cadena (`File.ReadAllText`)
    - Ejercicio 9: Guardar un array de strings en un archivo (`File.WriteAllLines`)
    - Ejercicio 10: Leer todas las líneas en un array y mostrarlas (`File.ReadAllLines`)

- Ejercicio 11: Crear un archivo binario con números enteros (`BinaryWriter`)
- Ejercicio 12: Leer números enteros de un archivo binario (`BinaryReader`)
- Ejercicio 13: Guardar una lista de objetos en archivo texto (`Tuple`, `StreamWriter`)
- Ejercicio 14: Leer archivo de personas y mostrar la más joven
- Ejercicio 15: Guardar y leer datos usando FileStream (bytes)
- Ejercicio 16: Crear un archivo CSV con encabezados (`StreamWriter`)
- Ejercicio 17: Leer archivo CSV y calcular total
- Ejercicio 18: Actualizar línea en un archivo de texto
- Ejercicio 19: Borrar línea que contiene un texto en archivo (`.Where()`, `.ToList()`)
- Ejercicio 20: Manejo de archivos con diálogo para seleccionar archivo (`OpenFileDialog`, `File.ReadAllText`) (Para Windows Forms)

- **30. Ejercicios de Archivos de Ficha** (Nivel Básico)

    1. Crear una ficha de agenda con nombre y teléfono
    2. Validar que el teléfono tenga solo números (`IsNumeric`)
    3. Guardar múltiples fichas en un arreglo
    4. Mostrar ficha por índice
    5. Usar estructura para representar fichas (`Structure`)

- **31. Controles de Interfaz Gráfica en Windows Forms – Visual Basic .NET**

    - Objetivos de la lección
        1. Introducción a Windows Forms
        2. Controles Básicos y su Uso
        3. 2.1 Label (Etiqueta)
        4. 2.2 TextBox (Cuadro de texto)
        5. 2.3 Button (Botón)
        6. 2.4 CheckBox (Casilla de verificación)
        7. 2.5 RadioButton (Botón de opción)
        8. 2.6 ComboBox (Lista desplegable)
        9. 2.7 ListBox (Lista de elementos)
        10. 2.8 PictureBox (Imagen)
        11. 2.9 DateTimePicker (Selector de fecha)
        12. 2.10 Timer (Temporizador)
        3. Ejemplo Práctico: Formulario de Registro
        4. Buenas Prácticas
        5. Actividades Propuestas
        6. Resumen

- **32. Ejercicios Form**

    - **Programas de dificultad fácil (1–20)**
        1. Mostrar mensaje al hacer clic en un botón
        2. Copiar texto de TextBox a Label
        3. Suma de dos números
        4. Cambiar color de fondo
        5. Mostrar opción de CheckBox

6. Seleccionar género con RadioButton
7. Mostrar fecha y hora actual
8. Agregar ítem a ListBox
9. Mostrar elemento seleccionado en ComboBox
10. Limpiar campos
11. Contador de clics
12. Cambiar texto de Label con ComboBox
13. Calcular área de un rectángulo
14. Activar/Desactivar botón con CheckBox
15. Concatenar nombre y apellido
16. Cambiar tamaño de fuente
17. Mostrar número aleatorio
18. Calcular el cuadrado de un número
19. Convertir texto a mayúsculas
20. Mostrar cantidad de caracteres

- **Programas de dificultad media (21–40)**
  21. Validar entrada numérica en TextBox
  22. Mostrar lista ordenada en ListBox
  23. Contar palabras en TextBox
  24. Mostrar imagen en PictureBox desde archivo (`OpenFileDialog`)
  25. Habilitar/Deshabilitar controles con RadioButtons
  26. Mostrar ítems seleccionados en ListBox múltiple
  27. Calcular IMC (Índice de Masa Corporal)
  28. Copiar texto con formato RichTextBox a TextBox
  29. Agregar fecha actual a ListBox
  30. Mostrar índice del ítem seleccionado en ComboBox
  31. Calcular promedio de números ingresados en TextBoxes
  32. Cambiar color de texto con ColorDialog
  33. Validar que TextBox no esté vacío antes de continuar (`String.IsNullOrWhiteSpace`)
  34. Añadir elementos de array a ListBox
  35. Seleccionar todo el texto de un TextBox con botón
  36. Convertir texto a minúsculas
  37. Mostrar posición del cursor en TextBox
  38. Contar elementos en ListBox
  39. Eliminar elemento seleccionado en ListBox
  40. Mostrar texto modificado en RichTextBox (`.Replace()`)
- **Programas de dificultad avanzado (41–60)**
  41. Temporizador que actualiza Label cada segundo (`Timer`)
  42. Validar correo electrónico con expresión regular (`System.Text.RegularExpressions.Regex`)
  43. Arrastrar y soltar texto entre TextBoxes (DragDrop)
  44. Graficar valores en un Chart (`DataVisualization.Charting.SeriesChartType.Column`)

45. Crear formulario modal y mostrar datos
46. Guardar texto de RichTextBox a archivo (`SaveFileDialog`, `System.IO.File.WriteAllText`)
47. Leer archivo de texto y mostrar en RichTextBox (`OpenFileDialog`, `System.IO.File.ReadAllText`)
48. Validar contraseña y mostrar mensaje
49. Mostrar progreso con ProgressBar (`Application.DoEvents()`, `Threading.Thread.Sleep()`)
50. Buscar texto en RichTextBox y resaltarlo (`RichTextBox.Find()`, `RichTextBox.Select()`, `RichTextBox.SelectionBackColor`)
51. Cambiar tamaño de ventana con TrackBar
52. Mostrar ToolTip en botón
53. Ordenar ListView por columna (`ListViewItemSorter`, `IComparer`)
54. Bloquear TextBox tras 3 intentos fallidos
55. Mostrar imagen desde recursos y cambiar con ComboBox (`My.Resources`)
56. Manejar evento DragDrop para archivos en formulario
57. Mostrar diálogo de impresión (`PrintDialog`)
58. Crear menú contextual para TextBox (`ContextMenuStrip`, `Clipboard.SetText`, `TextBox.Paste()`)
59. Mostrar formulario hijo dentro de formulario padre (MDI)
60. Guardar configuración usando My.Settings

- **33. Controles Visual Basic .NET** (Esta sección aparece dos veces, con el mismo contenido que "24. Controles Comunes...")
    - Controles Básicos (Tabla)
    - Controles Intermedios (Tabla)
    - Controles Avanzados (Tabla)
- **Apéndice**
    - Limpiar la pantalla en aplicaciones de consola (`Console.Clear()`)
    - Limpiar la pantalla en Windows Forms (TextBox, Label, recorrer controles)
    - Ejecutar un .exe externo (`System.Diagnostics.Process.Start`)
    - Interactuar entre formularios (Show, ShowDialog, Close)
    - Abrir un nuevo formulario desde otro
    - Cerrar el formulario actual (`Me.Close()`)
    - Abrir un formulario y cerrar el actual al mismo tiempo
    - Cambiar el comportamiento del cierre del formulario principal
    - Alternar entre formularios desde botones (Hide, Show)
    - Ejemplo completo paso a paso de alternar formularios en Windows Forms
- **Epílogo**

1. Fundamentos de Visual Basic .NET

1. Introducción

Visual Basic .NET (VB.NET) es un lenguaje de programación orientado a objetos desarrollado por Microsoft, que forma parte del entorno de desarrollo .NET. VB.NET permite crear aplicaciones de consola, de escritorio (Windows Forms), web (ASP.NET), servicios y más. Está diseñado para ser fácil de aprender y utilizar, especialmente para quienes comienzan en la programación.

---

2. Estructura básica de un programa en VB.NET

```
Module Module1
    Sub Main()
        Console.WriteLine("¡Hola, mundo!")
    End Sub
End Module
```

Explicación:

- `Module`: Un contenedor de código.
- `Sub Main()`: Punto de entrada de la aplicación.
- `Console.WriteLine`: Imprime texto en la consola.

---

3. Variables y tipos de datos

```
Dim edad As Integer = 25
Dim nombre As String = "Ana"
Dim precio As Double = 9.99
Dim activo As Boolean = True
```

Tipos comunes:

- `Integer`: Enteros.
- `Double`: Decimales.
- `String`: Cadenas de texto.
- `Boolean`: Verdadero/Falso.
- `Date`: Fechas.

---

4. Estructuras de control

Condicionales:

```
If edad >= 18 Then
    Console.WriteLine("Mayor de edad")
Else
    Console.WriteLine("Menor de edad")
End If
```

Select Case:

```
Select Case dia
    Case "Lunes"
        Console.WriteLine("Inicio de semana")
    Case "Viernes"
        Console.WriteLine("Casi fin de semana")
    Case Else
        Console.WriteLine("Otro día")
End Select
```

Bucles:

```
For i As Integer = 1 To 5
    Console.WriteLine("Número " & i)
Next

Dim i As Integer = 0
While i < 5
    Console.WriteLine("Contando " & i)
    i += 1
End While
```

---

5. Funciones y procedimientos

```
Function Sumar(a As Integer, b As Integer) As Integer
    Return a + b
End Function

Sub MostrarMensaje()
    Console.WriteLine("Bienvenido a VB.NET")
End Sub
```

---

6. Programación orientada a objetos

```
Class Persona
    Public Nombre As String
    Public Edad As Integer

    Public Sub Saludar()
        Console.WriteLine("Hola, soy " & Nombre)
    End Sub
End Class

Dim p As New Persona()
p.Nombre = "Carlos"
p.Edad = 30
p.Saludar()
```

---

7. Manejo de errores

```
Try
    Dim x As Integer = 10 / 0
Catch ex As Exception
    Console.WriteLine("Ocurrió un error: " & ex.Message)
Finally
    Console.WriteLine("Fin del bloque Try")
End Try
```

---

8. Entrada y salida

```
Console.Write("Introduce tu nombre: ")
Dim nombre As String = Console.ReadLine()
Console.WriteLine("Hola, " & nombre)
```

---

9. Comentarios en el código

```
' Esto es un comentario de una línea
```

---

10. Conclusión

Visual Basic .NET es un lenguaje potente y accesible para el desarrollo de todo tipo de aplicaciones en la plataforma .NET. Aprender sus fundamentos proporciona una base sólida para avanzar hacia proyectos más complejos y trabajar con tecnologías modernas como bases de datos, interfaces gráficas o servicios web.

---

2. Primeros Pasos con Visual Basic .NET

¿Qué es Visual Basic .NET?

Visual Basic .NET (VB.NET) es un lenguaje de programación orientado a objetos desarrollado por Microsoft. Forma parte de la plataforma .NET y es ampliamente utilizado para crear aplicaciones de escritorio, web y móviles, especialmente en entornos Windows.

---

1. Requisitos para comenzar

Antes de empezar a programar en VB.NET necesitas:

- **Visual Studio**: Entorno de desarrollo (IDE) recomendado por Microsoft.
- **.NET Framework / .NET Core / .NET 5+**: Plataforma para ejecutar tus programas.
- Sistema operativo **Windows** (opcional en caso de usar .NET Core multiplataforma).

Instalación básica:

1. Descarga e instala **Visual Studio Community** desde: https://visualstudio.microsoft.com/es/
2. Durante la instalación, selecciona el paquete "**Desarrollo de escritorio con .NET**".
3. Una vez instalado, abre Visual Studio y crea un nuevo proyecto.

---

2. Crear tu primer proyecto en VB.NET

Pasos para crear una aplicación de consola:

1. Abre Visual Studio.
2. Haz clic en **"Crear un nuevo proyecto"**.
3. Elige **"Aplicación de consola (.NET Core)"** o **"Aplicación de consola (.NET Framework)"** con Visual Basic.
4. Asigna un nombre al proyecto.
5. Haz clic en **"Crear"**.

---

3. Estructura básica de un programa en VB.NET

```
Module Module1

    Sub Main()
        Console.WriteLine("¡Hola, mundo!")
        Console.ReadLine()
    End Sub

End Module
```

Explicación:

- `Module Module1`: Un módulo es una agrupación de código.
- `Sub Main()`: Punto de entrada principal de la aplicación.
- `Console.WriteLine`: Imprime texto en la consola.
- `Console.ReadLine`: Espera a que el usuario presione Enter.

---

4. Variables y tipos de datos

```
Dim nombre As String = "Ana"
Dim edad As Integer = 30
Dim altura As Double = 1.65
Dim activo As Boolean = True
```

Principales tipos de datos:

- `String`: Texto
- `Integer`: Entero
- `Double`: Número decimal
- `Boolean`: Verdadero/Falso

---

5. Estructuras de control

Condicional If:

```
If edad >= 18 Then
    Console.WriteLine("Eres mayor de edad.")
Else
    Console.WriteLine("Eres menor de edad.")
End If
```

Bucle For:

```
For i As Integer = 1 To 5
    Console.WriteLine("Número " & i)
Next
```

Bucle While:

```
Dim i As Integer = 0
While i < 3
    Console.WriteLine("Repetición " & i)
    i += 1
End While
```

---

6. Crear funciones

```
Function Sumar(a As Integer, b As Integer) As Integer
    Return a + b
End Function

Sub Main()
    Dim resultado As Integer = Sumar(5, 7)
    Console.WriteLine("Resultado: " & resultado)
End Sub
```

---

7. Manejo de errores

```
Try
    Dim x As Integer = 5
    Dim y As Integer = 0
    Dim z As Integer = x / y
Catch ex As DivideByZeroException
    Console.WriteLine("Error: División por cero.")
Finally
    Console.WriteLine("Fin del proceso.")
End Try
```

---

8. Entrada del usuario

```
Console.Write("Introduce tu nombre: ")
Dim nombre As String = Console.ReadLine()
Console.WriteLine("Hola, " & nombre)
```

Conclusión

Visual Basic .NET es un lenguaje sencillo de aprender, especialmente para quienes empiezan en la programación. Con Visual Studio, puedes desarrollar aplicaciones completas utilizando herramientas gráficas y un entorno muy amigable.

---

3. Tipos de Datos y Variables en Visual Basic .NET

Visual Basic .NET (VB.NET) es un lenguaje de programación desarrollado por Microsoft que se ejecuta en la plataforma .NET. Uno de los fundamentos de la programación en VB.NET es la correcta declaración y uso de **variables** y **tipos de datos**.

---

1. ¿Qué es una variable?

Una **variable** es un espacio en memoria reservado para almacenar datos que pueden cambiar durante la ejecución de un programa. En VB.NET, las variables deben declararse antes de ser usadas.

Declaración de variables:

```
Dim nombre As String
Dim edad As Integer
Dim precio As Double
```

---

2. Reglas para nombrar variables

- Deben comenzar con una letra.
- No pueden contener espacios.
- No deben usar palabras clave del lenguaje.
- Se recomienda usar nombres descriptivos.

---

3. Tipos de datos en VB.NET

VB.NET es un lenguaje fuertemente tipado, lo que significa que cada variable debe tener un tipo de dato definido. A continuación se describen los tipos de datos más comunes:

Tipo de Dato	Tamaño	Rango o Descripción	Ejemplo
`Byte`	1 byte	0 a 255	`Dim b As Byte = 100`
`Short`	2 bytes	-32,768 a 32,767	`Dim s As Short = 15000`
`Integer`	4 bytes	-2,147,483,648 a 2,147,483,647	`Dim i As Integer = 500000`
`Long`	8 bytes	Muy grande, para números enteros largos	`Dim l As Long = 9223372036854775807`
`Single`	4 bytes	Números decimales con precisión simple	`Dim x As Single = 12.34`
`Double`	8 bytes	Números decimales con precisión doble	`Dim d As Double = 123456.789`
`Decimal`	16 bytes	Muy preciso, ideal para cálculos financieros	`Dim m As Decimal = 1000000.55D`
`Char`	2 bytes	Un solo carácter	`Dim c As Char = "A"`
`String`	Variable	Cadena de caracteres	`Dim str As String = "Hola Mundo"`
`Boolean`	1 byte	Verdadero o Falso (`True` o `False`)	`Dim b As Boolean = True`
`Date`	8 bytes	Representa fechas y horas	`Dim f As Date = #06/25/2025#`
`Object`	Variable	Puede almacenar cualquier tipo de dato	`Dim obj As Object = "texto"`

---

4. Conversión de tipos

VB.NET permite convertir tipos de datos mediante funciones:

```
Dim numero As Integer = CInt("123")
Dim texto As String = CStr(456)
Dim decimal As Decimal = CDec("45.67")
```

---

5. Inferencia de tipos (con `Option Infer`)

Con `Option Infer On`, VB.NET puede inferir automáticamente el tipo de una variable:

```
Option Infer On
Dim edad = 25        ' Se infiere como Integer
Dim nombre = "Ana"   ' Se infiere como String
```

---

6. Constantes

Para declarar constantes (valores que no cambian):

```
Const PI As Double = 3.14159
Const IVA As Decimal = 0.21D
```

---

7. Ámbito de las variables

El **ámbito** define dónde puede ser usada una variable:

- **Local**: declarada dentro de un procedimiento.
- **De clase (Private/Public)**: declarada dentro de una clase.
- **Global**: accesible desde varios módulos (con `Public` en un módulo).

---

8. Buenas prácticas

- Usa nombres descriptivos.
- Declara variables lo más cerca posible de donde se usen.
- Elige el tipo de dato más adecuado según el uso.
- Inicializa variables antes de usarlas.

---

4. Tipos de Operadores y Expresiones en Visual Basic .NET

1. ¿Qué es una Expresión?

Una **expresión** es una combinación de valores, variables, operadores y llamadas a funciones que se pueden evaluar y devuelven un resultado. Por ejemplo:

```
Dim resultado As Integer = 5 + 3
```

La expresión `5 + 3` se evalúa y devuelve el valor `8`.

---

2. Tipos de Operadores en Visual Basic .NET

Visual Basic .NET proporciona varios tipos de operadores que se pueden usar en expresiones:

2.1 Operadores Aritméticos

Se usan para realizar operaciones matemáticas básicas.

Operador	Descripción	Ejemplo
`+`	Suma	`a + b`
`-`	Resta	`a - b`
`*`	Multiplicación	`a * b`
`/`	División (decimal)	`a / b`
`\`	División entera	`a \ b`
`Mod`	Módulo (resto)	`a Mod b`
`^`	Potencia	`a ^ b`

Ejemplo:

```
Dim x As Integer = 10
Dim y As Integer = 3
Dim resultado As Integer = x Mod y ' resultado = 1
```

2.2 Operadores de Comparación

Se utilizan para comparar valores.

Operador	Descripción	Ejemplo
`=`	Igual a	`a = b`
`<>`	Distinto de	`a <> b`
`>`	Mayor que	`a > b`
`<`	Menor que	`a < b`
`>=`	Mayor o igual que	`a >= b`
`<=`	Menor o igual que	`a <= b`
`Is`	Igualdad de objetos	`obj1 Is obj2`

Ejemplo:

```
If edad >= 18 Then
    Console.WriteLine("Es mayor de edad")
End If
```

2.3 Operadores Lógicos

Permiten realizar operaciones lógicas sobre valores booleanos.

Operador	Descripción	Ejemplo
`And`	Conjunción	`a And b`
`Or`	Disyunción	`a Or b`
`Not`	Negación	`Not a`
`AndAlso`	Conjunción corta	`a AndAlso b`
`OrElse`	Disyunción corta	`a OrElse b`
`Xor`	Disyunción exclusiva	`a Xor b`

Ejemplo:

```
Dim accesoPermitido As Boolean = (usuario = "admin") AndAlso (clave = "1234")
```

2.4 Operadores de Asignación

Sirven para asignar valores a variables.

Operador	Descripción	Ejemplo
`=`	Asignación simple	`x = 10`
`+=`	Suma y asigna	`x += 5`
`-=`	Resta y asigna	`x -= 3`
`*=`	Multiplica y asigna	`x *= 2`
`/=`	Divide y asigna	`x /= 4`
`\=`	División entera y asigna	`x \= 2`
`^=`	Potencia y asigna	`x ^= 3`
`&=`	Concatena y asigna	`cadena &= "texto"`

2.5 Operadores de Concatenación

Usados para unir cadenas de texto.

Operador	Descripción	Ejemplo
`&`	Concatenación	`"Hola" & " Mundo"`
`+`	También concatena, pero puede causar confusión si hay números	

Ejemplo:

```
Dim nombre As String = "Ana"
Dim saludo As String = "Hola, " & nombre
```

2.6 Operadores de Bits

Se usan para realizar operaciones a nivel de bits sobre enteros.

Operador	Descripción	Ejemplo
`And`	AND bit a bit	`a And b`
`Or`	OR bit a bit	`a Or b`
`Xor`	XOR bit a bit	`a Xor b`
`Not`	NOT bit a bit	`Not a`
`<<`	Desplazamiento izq.	`a << 1`
`>>`	Desplazamiento der.	`a >> 2`

---

3. Precedencia de Operadores

La precedencia determina el orden en que se evalúan los operadores. Por ejemplo:

```
Dim resultado As Integer = 5 + 2 * 3 ' resultado = 11
```

Se evalúa primero la multiplicación. Puedes usar paréntesis para modificar el orden:

```
Dim resultado As Integer = (5 + 2) * 3 ' resultado = 21
```

---

4. Expresiones Compuestas

Una expresión puede combinar distintos tipos de operadores:

```
If (edad >= 18 And pais = "España") Or esResidente Then
    Console.WriteLine("Acceso concedido")
End If
```

Conclusión

Los operadores y expresiones son esenciales en Visual Basic .NET para manipular datos, tomar decisiones, realizar cálculos y controlar el flujo de un programa. Conocer su sintaxis y precedencia permite escribir código más preciso y eficiente.

---

5. estructuras de control en Visual Basic .NET,

---

1. Introducción

Las **estructuras de control** permiten alterar el flujo normal de ejecución de un programa. En Visual Basic .NET, estas estructuras se dividen principalmente en:

- Estructuras condicionales (decisiones)
- Estructuras repetitivas (bucles o ciclos)
- Estructuras de selección múltiple

---

2. Estructuras Condicionales

2.1 If...Then

Se utiliza para ejecutar un bloque de código si una condición se cumple.

```
Dim edad As Integer = 20

If edad >= 18 Then
    Console.WriteLine("Eres mayor de edad.")
End If
```

2.2 If...Then...Else

Permite ejecutar diferentes bloques según se cumpla o no la condición.

```
If edad >= 18 Then
    Console.WriteLine("Mayor de edad")
Else
    Console.WriteLine("Menor de edad")
End If
```

2.3 If...Then...ElseIf

Permite múltiples condiciones.

```
If edad < 13 Then
    Console.WriteLine("Niño")
ElseIf edad < 18 Then
    Console.WriteLine("Adolescente")
Else
    Console.WriteLine("Adulto")
End If
```

2.4 Operador Ternario (IIf o If Inline)

```
Dim mensaje As String = If(edad >= 18, "Mayor", "Menor")
Console.WriteLine(mensaje)
```

---

3. Estructuras de Selección Múltiple

3.1 Select Case

Útil cuando se evalúa una misma variable con varios posibles valores.

```
Dim dia As Integer = 3

Select Case dia
    Case 1
        Console.WriteLine("Lunes")
    Case 2
        Console.WriteLine("Martes")
    Case 3
        Console.WriteLine("Miércoles")
    Case Else
        Console.WriteLine("Otro día")
End Select
```

---

4. Estructuras Repetitivas (Bucles)

4.1 For...Next

Cuando se conoce el número de repeticiones.

```
For i As Integer = 1 To 5
    Console.WriteLine("Iteración " & i)
Next
```

4.2 For...Each

Para recorrer colecciones o arrays.

```
Dim frutas() As String = {"Manzana", "Banana", "Cereza"}

For Each fruta As String In frutas
    Console.WriteLine(fruta)
Next
```

4.3 While...End While

Repite mientras la condición sea verdadera.

```
Dim contador As Integer = 0

While contador < 5
    Console.WriteLine("Contador: " & contador)
    contador += 1
End While
```

4.4 Do While / Do Until

Se ejecuta mientras (o hasta que) la condición se cumpla.

```
Dim n As Integer = 0

Do While n < 3
    Console.WriteLine("Valor de n: " & n)
    n += 1
Loop
```

También puede usarse `Do...Loop Until` o `Do...Loop While` después del cuerpo del bucle.

---

5. Salida Anticipada de Bucles

5.1 Exit For / Exit Do

Finaliza un bucle antes de que termine normalmente.

```
For i As Integer = 1 To 10
    If i = 5 Then
        Exit For
    End If
    Console.WriteLine(i)
Next
```

5.2 Continue For / Continue Do

Salta a la siguiente iteración del bucle.

```
For i As Integer = 1 To 5
    If i = 3 Then
        Continue For
    End If
    Console.WriteLine(i)
Next
```

---

6. Conclusión

Las estructuras de control en VB.NET son esenciales para desarrollar aplicaciones con lógica flexible. Dominar estas estructuras permite crear programas dinámicos, eficientes y adaptativos.

---

6. Procedimientos y Funciones en Visual Basic .NET

1. Introducción

En Visual Basic .NET, los procedimientos y funciones son bloques de código reutilizables que realizan una tarea específica. Permiten dividir un programa en partes lógicas, facilitando su lectura, mantenimiento y reutilización.

---

2. Tipos de procedimientos

2.1 Subprocedimientos (`Sub`)

Un `Sub` es un procedimiento que ejecuta código pero **no devuelve un valor**.

Sintaxis:

```
Sub NombreProcedimiento()
    ' Instrucciones
End Sub
```

Ejemplo:

```
Sub MostrarMensaje()
    Console.WriteLine("Hola, mundo")
End Sub
```

Uso:

```
MostrarMensaje()
```

2.2 Procedimientos con parámetros

Se pueden pasar datos a los procedimientos usando parámetros.

```
Sub Saludar(nombre As String)
    Console.WriteLine("Hola, " & nombre)
End Sub

' Uso
Saludar("Ana")
```

---

3. Funciones (`Function`)

Una `Function` es similar a un `Sub`, pero **devuelve un valor** al llamarla.

Sintaxis:

```
Function NombreFuncion() As TipoDeDato
    ' Instrucciones
    Return valor
End Function
```

Ejemplo:

```
Function ObtenerSuma(a As Integer, b As Integer) As Integer
    Return a + b
```

```
End Function

' Uso
Dim resultado As Integer = ObtenerSuma(5, 3)
Console.WriteLine(resultado)  ' Imprime 8
```

---

4. Parámetros por valor y por referencia

- **ByVal** (por defecto): se pasa una copia del valor.
- **ByRef**: se pasa una referencia, por lo que el valor puede cambiar dentro del procedimiento.

Ejemplo de ByRef:

```
Sub Incrementar(ByRef numero As Integer)
    numero += 1
End Sub

Dim x As Integer = 5
Incrementar(x)
Console.WriteLine(x)  ' Imprime 6
```

---

5. Funciones con valores por defecto

Puedes asignar un valor predeterminado a un parámetro:

```
Sub Saludo(nombre As String, Optional mensaje As String = "Hola")
    Console.WriteLine(mensaje & ", " & nombre)
End Sub

Saludo("Luis")                    ' Imprime: Hola, Luis
Saludo("Luis", "Buenos días")     ' Imprime: Buenos días, Luis
```

---

6. Funciones anidadas

Visual Basic .NET permite definir funciones dentro de otras funciones (local functions), a partir de VB 15.0 (en .NET Core).

```
Function CalcularCuadrado(valor As Integer) As Integer
    Dim Cuadrado = Function(x As Integer) x * x
    Return Cuadrado(valor)
End Function
```

---

7. Buenas prácticas

- Usa nombres descriptivos para funciones y procedimientos.
- Evita funciones muy largas: divide en funciones más pequeñas.
- Documenta con comentarios.
- Usa `Function` solo si necesitas devolver un valor.

---

8. Ejercicio práctico

Objetivo: Crear un programa que calcule el área de un rectángulo y muestre el resultado.

```
Module Program
    Sub Main()
        Dim ancho As Double = 5.5
        Dim alto As Double = 3.2
        Dim area As Double = CalcularArea(ancho, alto)
        MostrarResultado(area)
    End Sub

    Function CalcularArea(ancho As Double, alto As Double) As Double
        Return ancho * alto
    End Function

    Sub MostrarResultado(area As Double)
        Console.WriteLine("El área es: " & area)
    End Sub
End Module
```

---

7. Arreglos y Colecciones en Visual Basic .NET

1. ¿Qué es un Arreglo?

Un **arreglo** (array) es una estructura de datos que almacena una secuencia de elementos del mismo tipo.

Declaración y uso básico:

```
Dim numeros(4) As Integer ' Arreglo de 5 enteros (de 0 a 4)

numeros(0) = 10
numeros(1) = 20
numeros(2) = 30
numeros(3) = 40
numeros(4) = 50

For i As Integer = 0 To numeros.Length - 1
    Console.WriteLine(numeros(i))
Next
```

Arreglos multidimensionales:

```
Dim matriz(2, 2) As Integer ' Arreglo bidimensional 3x3

matriz(0, 0) = 1
matriz(0, 1) = 2
matriz(0, 2) = 3
' ...
```

Arreglos dinámicos (redimensionables):

```
Dim valores() As Integer
ReDim valores(2) ' Crea arreglo de 3 elementos

valores(0) = 5
valores(1) = 10
valores(2) = 15

ReDim Preserve valores(4) ' Redimensiona sin perder datos
valores(3) = 20
valores(4) = 25
```

---

2. ¿Qué es una Colección?

Una **colección** es una clase que puede contener múltiples elementos, pero a diferencia de los arreglos, puede crecer o decrecer dinámicamente. Las colecciones permiten trabajar con diferentes tipos de datos, ordenarlos, buscarlos, etc.

2.1. `ArrayList`

```
Imports System.Collections

Dim lista As New ArrayList()

lista.Add("Manzana")
lista.Add("Banana")
lista.Add(100)

For Each item In lista
    Console.WriteLine(item)
Next
```

> ⚠ `ArrayList` permite mezclar tipos de datos, pero no es tipo seguro.

2.2. `List(Of T)` (Genérica y tipo seguro)

```
Dim numeros As New List(Of Integer)

numeros.Add(10)
numeros.Add(20)
numeros.Add(30)

For Each numero In numeros
    Console.WriteLine(numero)
Next
```

2.3. `Dictionary(Of TKey, TValue)`

Permite almacenar pares clave-valor.

```
Dim diccionario As New Dictionary(Of String, String)

diccionario.Add("ES", "España")
diccionario.Add("FR", "Francia")
diccionario.Add("IT", "Italia")

Console.WriteLine(diccionario("FR")) ' Devuelve "Francia"
```

---

3. Comparación entre Arreglos y Colecciones

Característica	Arreglos	Colecciones (List, Dictionary)
Tamaño	Fijo (o ReDim)	Dinámico
Tipado	Fuerte	Fuerte (en genéricos)
Rendimiento	Más rápido en acceso directo	Más flexible, pero ligeramente más lento
Funciones integradas	Limitadas	Amplias (Add, Remove, Find, etc.)

---

4. Ejemplo práctico con `List(Of String)`

```
Dim nombres As New List(Of String) From {"Ana", "Luis", "Carlos"}

nombres.Add("Marta")
nombres.Remove("Luis")

If nombres.Contains("Carlos") Then
    Console.WriteLine("Carlos está en la lista")
End If

nombres.Sort()
For Each nombre In nombres
    Console.WriteLine(nombre)
Next
```

---

5. Conclusión

En Visual Basic .NET, tanto los arreglos como las colecciones son herramientas fundamentales para manejar conjuntos de datos.

- Usa **arreglos** cuando sepas de antemano la cantidad exacta de elementos.
- Usa **colecciones** cuando necesites más flexibilidad o trabajar con datos de forma más compleja.

---

8. Manejo de cadenas de texto en Visual Basic .NET

---

1. Introducción

Las **cadenas de texto** (tipo `String`) son fundamentales en cualquier lenguaje de programación. En Visual Basic .NET (VB.NET), trabajar con cadenas es muy común para tareas como entrada/salida de datos, validaciones, formateo, búsqueda y reemplazo de texto.

---

2. Declaración de cadenas

```
Dim saludo As String = "Hola, mundo"
```

También se puede declarar vacía:

```
Dim cadenaVacia As String = ""
```

---

3. Concatenación de cadenas

Se pueden unir varias cadenas usando el operador & o +:

```
Dim nombre As String = "Ana"
Dim mensaje As String = "Hola, " & nombre
Console.WriteLine(mensaje)  ' Salida: Hola, Ana
```

---

4. Longitud de una cadena

```
Dim texto As String = "Visual Basic"
Console.WriteLine(texto.Length)  ' Salida: 12
```

---

5. Acceder a caracteres individuales

```
Dim letra As Char = texto(0)  ' Primer carácter
Console.WriteLine(letra)  ' Salida: V
```

---

6. Subcadenas

```
Dim subcadena As String = texto.Substring(0, 6)
Console.WriteLine(subcadena)  ' Salida: Visual
```

---

7. Conversión a mayúsculas y minúsculas

```
Console.WriteLine(texto.ToUpper())  ' VISUAL BASIC
Console.WriteLine(texto.ToLower())  ' visual basic
```

8. Búsqueda de texto

```
Dim indice As Integer = texto.IndexOf("Basic")
Console.WriteLine(indice)  ' Salida: 7
```

Si no se encuentra, devuelve -1.

---

9. Reemplazo de texto

```
Dim nuevaCadena As String = texto.Replace("Basic", "Studio")
Console.WriteLine(nuevaCadena)  ' Visual Studio
```

---

10. Eliminación de espacios

```
Dim textoEspaciado As String = "   Hola   "
Console.WriteLine(">" & textoEspaciado.Trim() & "<")  ' >Hola<
```

- `Trim()`: elimina espacios al inicio y final.
- `TrimStart()`: solo al inicio.
- `TrimEnd()`: solo al final.

---

11. Comparación de cadenas

```
Dim a As String = "casa"
Dim b As String = "CASA"
Console.WriteLine(a = b)  ' False (por defecto es sensible a mayúsculas)

' Comparación sin distinguir mayúsculas:
Console.WriteLine(String.Compare(a, b, True))  ' 0 (son iguales)
```

---

12. Dividir una cadena

```
Dim frase As String = "uno,dos,tres"
Dim partes() As String = frase.Split(","c)
For Each parte In partes
    Console.WriteLine(parte)
Next
```

---

13. Unir elementos en una cadena

```
Dim palabras() As String = {"rojo", "verde", "azul"}
Dim resultado As String = String.Join(" - ", palabras)
Console.WriteLine(resultado)  ' rojo - verde - azul
```

---

14. Verificar contenido

```
Console.WriteLine(texto.Contains("Basic"))  ' True
Console.WriteLine(texto.StartsWith("Visual"))  ' True
Console.WriteLine(texto.EndsWith("ic"))  ' True
```

---

15. Formateo de cadenas

```
Dim edad As Integer = 30
Dim nombre As String = "Luis"
Console.WriteLine(String.Format("Nombre: {0}, Edad: {1}", nombre, edad))
' Salida: Nombre: Luis, Edad: 30
```

También puedes usar interpolación de cadenas (VB.NET 14+):

```
Console.WriteLine($"Nombre: {nombre}, Edad: {edad}")
```

---

16. Ejemplo completo

```
Module Program
    Sub Main()
        Dim nombre As String = " Violeta "
        nombre = nombre.Trim()
        Dim saludo As String = $"Hola, {nombre.ToUpper()}!"
        Console.WriteLine(saludo)

        If saludo.Contains("VIOLETA") Then
            Console.WriteLine("El nombre está presente.")
        End If
    End Sub
End Module
```

---

17. Conclusión

El manejo de cadenas en VB.NET es sencillo gracias a la gran cantidad de métodos disponibles. Dominar estas funciones te permitirá realizar validaciones, análisis de texto, formateos y más de forma eficiente.

---

9. Clases y Objetos en Visual Basic .NET

Introducción

En la programación orientada a objetos (POO), una **clase** es una plantilla o modelo que define las propiedades y comportamientos de un objeto. Un **objeto** es una instancia de una clase: algo concreto que existe en la memoria durante la ejecución de un programa.

Visual Basic .NET (VB.NET) es un lenguaje totalmente orientado a objetos, lo que significa que todo —incluidos los formularios y controles— se basa en clases y objetos.

---

1. Definición de una Clase

Una clase en VB.NET se define con la palabra clave `Class`. Dentro de ella se pueden incluir propiedades, campos, métodos y constructores.

```
Public Class Persona
    ' Campo privado
    Private nombre As String

    ' Propiedad pública
    Public Property NombreCompleto As String
        Get
            Return nombre
        End Get
        Set(value As String)
            nombre = value
        End Set
    End Property

    ' Método
    Public Sub Saludar()
        Console.WriteLine("Hola, mi nombre es " & nombre)
    End Sub
End Class
```

---

2. Crear un Objeto (Instancia de una Clase)

Una vez definida la clase, se puede crear un objeto con la palabra clave `New`.

```
Module Program
    Sub Main()
        Dim persona1 As New Persona()
        persona1.NombreCompleto = "Ana Martínez"
        persona1.Saludar()
    End Sub
End Module
```

---

3. Constructores

Un **constructor** es un método especial que se ejecuta cuando se crea un objeto. Se define con `Sub New()`.

```
Public Class Persona
    Private nombre As String

    ' Constructor
    Public Sub New(nom As String)
        nombre = nom
    End Sub

    Public Sub Saludar()
        Console.WriteLine("Hola, soy " & nombre)
    End Sub
End Class

Module Program
    Sub Main()
        Dim persona2 As New Persona("Luis Gómez")
        persona2.Saludar()
    End Sub
End Module
```

---

4. Encapsulamiento

VB.NET permite ocultar los detalles internos de una clase usando modificadores como `Private` y exponer solo lo necesario con `Public` o `Friend`.

---

5. Herencia

Las clases pueden **heredar** de otras. Se usa la palabra clave `Inherits`.

```
Public Class Empleado
    Inherits Persona

    Public Puesto As String

    Public Sub MostrarPuesto()
        Console.WriteLine("Trabajo como " & Puesto)
    End Sub
End Class
```

---

6. Polimorfismo

El polimorfismo permite que métodos con el mismo nombre se comporten de forma diferente según el objeto.

```
Public Overridable Sub Saludar()
    Console.WriteLine("Hola desde Persona")
End Sub

Public Overrides Sub Saludar()
    Console.WriteLine("Hola desde Empleado")
End Sub
```

---

7. Clases y Métodos Compartidos (Shared)

Una clase o método puede ser compartido por todos los objetos sin necesidad de crear una instancia.

```
Public Class Utilidad
    Public Shared Sub MostrarMensaje()
        Console.WriteLine("Este es un mensaje compartido.")
    End Sub
End Class

' Uso
Utilidad.MostrarMensaje()
```

Conclusión

En VB.NET, las clases y objetos son fundamentales para organizar y reutilizar el código. Comprender sus elementos (propiedades, métodos, constructores, herencia y encapsulamiento) es esencial para desarrollar software robusto y modular.

---

10. Encapsulamiento, Herencia y Polimorfismo en Visual Basic .NET

1. Encapsulamiento

El **encapsulamiento** es el principio de ocultar los detalles internos de un objeto y exponer solo lo necesario. En Visual Basic .NET, esto se logra mediante **modificadores de acceso** (`Public`, `Private`, `Protected`, etc.) y propiedades (`Property`).

Ejemplo:

```
Public Class Persona
    Private nombre As String

    Public Property NombrePersona As String
        Get
            Return nombre
        End Get
        Set(value As String)
            If value <> "" Then
                nombre = value
            End If
        End Set
    End Property
End Class
```

En este ejemplo:

- El campo `nombre` está **encapsulado** como `Private`.
- Se accede a él mediante la propiedad pública `NombrePersona`, que permite validar los datos antes de asignarlos.

---

2. Herencia

La **herencia** permite crear nuevas clases basadas en clases existentes. La clase derivada hereda miembros de la clase base y puede agregar o modificar su funcionalidad.

Sintaxis:

```
Public Class Animal
    Public Overridable Sub Hablar()
        Console.WriteLine("Sonido genérico")
    End Sub
End Class

Public Class Perro
    Inherits Animal

    Public Overrides Sub Hablar()
        Console.WriteLine("Guau")
    End Sub
End Class
```

Notas:

- `Inherits` indica que una clase hereda de otra.
- `Overridable` en la clase base permite que un método sea sobrescrito.
- `Overrides` en la clase derivada redefine el comportamiento del método.

---

3. Polimorfismo

El **polimorfismo** permite que diferentes clases respondan de manera diferente al mismo método, usando la **misma interfaz**. Hay dos tipos principales:

- **Polimorfismo en tiempo de compilación (sobrecarga)**
- **Polimorfismo en tiempo de ejecución (sobrescritura)**

Ejemplo de sobrescritura (tiempo de ejecución):

```
Dim listaAnimales As New List(Of Animal)
listaAnimales.Add(New Perro())
listaAnimales.Add(New Gato())

For Each a As Animal In listaAnimales
    a.Hablar()
Next
```

Si `Perro` y `Gato` heredan de `Animal` y sobrescriben `Hablar()`, cada uno ejecutará su propia versión.

Ejemplo de sobrecarga (tiempo de compilación):

```
Public Sub MostrarMensaje(ByVal mensaje As String)
    Console.WriteLine(mensaje)
End Sub

Public Sub MostrarMensaje(ByVal numero As Integer)
    Console.WriteLine("Número: " & numero)
End Sub
```

Aquí, `MostrarMensaje` está sobrecargado con diferentes parámetros.

Conclusión

Visual Basic .NET ofrece una implementación completa de los principios fundamentales de la programación orientada a objetos:

- El **encapsulamiento** permite proteger y controlar el acceso a los datos.
- La **herencia** promueve la reutilización del código.
- El **polimorfismo** permite manejar objetos de diferentes tipos de manera uniforme.

Estos conceptos son clave para escribir código estructurado, reutilizable y mantenible en aplicaciones VB.NET.

11. Interfaces y Clases Abstractas en Visual Basic .NET

1. Introducción

En Visual Basic .NET, las interfaces y las clases abstractas son herramientas fundamentales para la programación orientada a objetos. Ambas permiten definir contratos y estructuras base para clases derivadas, promoviendo la reutilización de código y la implementación flexible.

---

2. ¿Qué es una clase abstracta?

Una **clase abstracta** es una clase que no se puede instanciar directamente. Se utiliza como una base para otras clases que deben heredarla y completar su funcionalidad.

Características:

- Puede contener métodos implementados y no implementados.
- Puede tener constructores, propiedades y campos.
- Se hereda con la palabra clave `Inherits`.

Sintaxis:

```
Public MustInherit Class Figura
    Public MustOverride Function CalcularArea() As Double
    Public Sub Mostrar()
        Console.WriteLine("Método común de todas las figuras")
    End Sub
End Class
```

Clase derivada:

```
Public Class Circulo
    Inherits Figura

    Public Property Radio As Double

    Public Overrides Function CalcularArea() As Double
        Return Math.PI * Radio * Radio
    End Function
End Class
```

---

3. ¿Qué es una interfaz?

Una **interfaz** es una colección de firmas de métodos (sin implementación) que una clase debe implementar completamente. Representa un contrato que garantiza que se implementarán ciertos métodos.

Características:

- No contiene lógica de implementación.
- No puede contener campos ni constructores.
- Se implementa con la palabra clave `Implements`.

Sintaxis:

```
Public Interface IImprimible
    Sub Imprimir()
End Interface
```

Clase que implementa la interfaz:

```
Public Class Documento
    Implements IImprimible

    Public Sub Imprimir() Implements IImprimible.Imprimir
        Console.WriteLine("Imprimiendo documento...")
    End Sub
End Class
```

---

4. Diferencias clave

Característica	Clase Abstracta	Interfaz
Instanciación	No se puede instanciar	No se puede instanciar
Implementación	Puede tener métodos implementados	No tiene implementación
Herencia múltiple	Solo una clase base	Se pueden implementar varias
Campos y constructores	Sí	No
Palabra clave	`MustInherit`, `MustOverride`	`Interface`, `Implements`

---

5. Cuándo usar una u otra

- Usa **interfaces** cuando necesitas asegurar que varias clases no relacionadas compartan métodos comunes (por ejemplo, `IDisposable`).
- Usa **clases abstractas** cuando tienes una jerarquía lógica de clases con funcionalidad compartida.

---

6. Ejemplo combinado

```
Public Interface IMensaje
    Sub Enviar()
End Interface

Public MustInherit Class Notificacion
    Public MustOverride Sub Mostrar()
End Class

Public Class Email
    Inherits Notificacion
    Implements IMensaje

    Public Overrides Sub Mostrar()
        Console.WriteLine("Mostrando Email")
    End Sub

    Public Sub Enviar() Implements IMensaje.Enviar
        Console.WriteLine("Enviando Email")
    End Sub
End Class
```

---

7. Conclusión

Interfaces y clases abstractas son elementos esenciales de la arquitectura de aplicaciones en Visual Basic .NET. Usarlas correctamente permite escribir código más limpio, reutilizable y flexible.

---

12. MANEJO DE ERRORES Y EXCEPCIONES EN VISUAL BASIC .NET

---

1. INTRODUCCIÓN

El manejo de errores es un aspecto fundamental de cualquier aplicación robusta. En Visual Basic .NET (VB.NET), el sistema de manejo de errores permite detectar, capturar y responder a condiciones inesperadas o excepcionales que ocurren durante la ejecución del programa.

---

2. TIPOS DE ERRORES

1. **Errores de compilación:** Detectados por el compilador antes de ejecutar el programa (por ejemplo, errores de sintaxis).
2. **Errores de ejecución:** Ocurren durante la ejecución del programa (por ejemplo, división por cero).
3. **Errores lógicos:** El programa se ejecuta sin fallos, pero no produce el resultado esperado.

---

3. ESTRUCTURA TRY...CATCH...FINALLY

La estructura principal para el manejo de errores en VB.NET es:

```
Try
    ' Código que puede generar una excepción
Catch ex As Exception
    ' Código para manejar la excepción
Finally
    ' Código que se ejecuta siempre, ocurra o no una excepción
End Try
```

Ejemplo básico:

```
Try
    Dim num1 As Integer = 10
    Dim num2 As Integer = 0
    Dim resultado As Integer = num1 \ num2
Catch ex As DivideByZeroException
    Console.WriteLine("No se puede dividir por cero.")
Catch ex As Exception
    Console.WriteLine("Se ha producido un error: " & ex.Message)
Finally
    Console.WriteLine("Proceso finalizado.")
End Try
```

---

4. OBJETO `Exception`

La clase base para todas las excepciones es `System.Exception`. Algunas propiedades útiles son:

- `Message`: Descripción del error.
- `StackTrace`: Ruta de llamadas que llevaron al error.
- `InnerException`: Excepción que causó la actual, si existe.
- `Source`: Nombre del ensamblado que generó la excepción.

---

5. JERARQUÍA DE EXCEPCIONES COMUNES

- `System.Exception`
    - `System.SystemException`
        - `DivideByZeroException`
        - `IndexOutOfRangeException`
        - `NullReferenceException`
        - `InvalidCastException`
        - `FormatException`
        - `IOException`, etc.

---

6. BLOQUE MULTIPLE CATCH

Puedes manejar distintos tipos de excepciones en bloques separados:

```
Try
    ' Código riesgoso
Catch ex As FormatException
    Console.WriteLine("Error de formato.")
Catch ex As OverflowException
    Console.WriteLine("Desbordamiento de número.")
Catch ex As Exception
    Console.WriteLine("Otro error: " & ex.Message)
End Try
```

---

7. BLOQUE FINALLY

El bloque `Finally` se ejecuta siempre, incluso si hay una instrucción `Exit Try` o si se lanza una excepción. Se usa para liberar recursos:

```
Dim archivo As IO.StreamReader = Nothing
Try
    archivo = New IO.StreamReader("datos.txt")
    Console.WriteLine(archivo.ReadToEnd())
Catch ex As IO.IOException
    Console.WriteLine("Error de archivo: " & ex.Message)
Finally
    If archivo IsNot Nothing Then archivo.Close()
End Try
```

---

8. LANZAMIENTO DE EXCEPCIONES

Puedes lanzar excepciones personalizadas usando la instrucción `Throw`:

```
Throw New Exception("Se produjo un error manual.")
```

Para relanzar la excepción actual en un `Catch`, se usa simplemente:

```
Throw
```

---

9. EXCEPCIONES PERSONALIZADAS

Puedes crear tus propias clases de excepción:

```
Public Class MiExcepcionPersonal
    Inherits Exception

    Public Sub New(ByVal mensaje As String)
        MyBase.New(mensaje)
    End Sub
End Class
```

Uso:

```
Throw New MiExcepcionPersonal("Este es un error definido por el usuario.")
```

---

10. MANEJO DE ERRORES CON `On Error` (NO RECOMENDADO)

Aunque VB.NET aún soporta el estilo antiguo de manejo de errores con `On Error GoTo`, **no se recomienda su uso** en programación moderna:

```
On Error GoTo ManejadorErrores
' Código principal
Exit Sub

ManejadorErrores:
    MsgBox("Error detectado: " & Err.Description)
```

---

11. BUENAS PRÁCTICAS

- Captura solo excepciones que puedas manejar.
- Usa `Finally` para liberar recursos como archivos o conexiones.
- No abuses de excepciones para controlar el flujo normal del programa.
- Registra errores para análisis posterior (logs).
- Usa excepciones personalizadas cuando necesites mayor precisión.

---

13. ACCESO A ARCHIVOS EN VISUAL BASIC .NET

---

1. Introducción

El acceso a archivos en Visual Basic .NET permite a las aplicaciones leer, escribir, modificar y borrar archivos de texto y binarios. Es una funcionalidad esencial para almacenar datos persistentes, como configuraciones, registros o documentos.

---

2. Espacio de nombres necesario

Para trabajar con archivos en VB.NET se utiliza principalmente el espacio de nombres:

```
Imports System.IO
```

---

3. Escribir en un archivo de texto

Ejemplo: Escribir texto en un archivo (sobrescribe el archivo si existe)

```
Imports System.IO

Module Module1
    Sub Main()
        Dim ruta As String = "C:\archivo.txt"
        File.WriteAllText(ruta, "Hola, este es un ejemplo de escritura.")
        Console.WriteLine("Archivo escrito correctamente.")
    End Sub
End Module
```

Ejemplo: Agregar texto a un archivo existente

```
Dim ruta As String = "C:\archivo.txt"
File.AppendAllText(ruta, Environment.NewLine & "Línea adicional.")
```

---

4. Leer un archivo de texto

Ejemplo: Leer todo el contenido de un archivo

```
Dim ruta As String = "C:\archivo.txt"
Dim contenido As String = File.ReadAllText(ruta)
Console.WriteLine("Contenido del archivo:")
Console.WriteLine(contenido)
```

Ejemplo: Leer línea por línea

```
Dim ruta As String = "C:\archivo.txt"
Dim lineas As String() = File.ReadAllLines(ruta)

For Each linea As String In lineas
    Console.WriteLine(linea)
Next
```

---

5. Leer y escribir usando `StreamReader` y `StreamWriter`

StreamWriter: Escribir en archivo

```
Using sw As New StreamWriter("C:\archivo.txt")
    sw.WriteLine("Primera línea")
    sw.WriteLine("Segunda línea")
End Using
```

StreamReader: Leer de archivo

```
Using sr As New StreamReader("C:\archivo.txt")
    Dim linea As String
    linea = sr.ReadLine()

    While linea IsNot Nothing
        Console.WriteLine(linea)
        linea = sr.ReadLine()
    End While
End Using
```

---

6. Verificar existencia y eliminar archivos

Comprobar si un archivo existe:

```
If File.Exists("C:\archivo.txt") Then
    Console.WriteLine("El archivo existe.")
Else
    Console.WriteLine("El archivo no existe.")
End If
```

Eliminar un archivo:

```
If File.Exists("C:\archivo.txt") Then
    File.Delete("C:\archivo.txt")
    Console.WriteLine("Archivo eliminado.")
End If
```

---

7. Crear y manejar directorios

```
Dim carpeta As String = "C:\NuevaCarpeta"

If Not Directory.Exists(carpeta) Then
    Directory.CreateDirectory(carpeta)
    Console.WriteLine("Carpeta creada.")
End If
```

---

8. Buenas prácticas

- Usa `Using` para asegurar el cierre correcto de archivos.
- Verifica si los archivos existen antes de abrirlos.
- Captura excepciones (`Try...Catch`) para manejar errores de acceso.
- Usa rutas relativas cuando sea posible para portabilidad.
- No bloquees archivos innecesariamente con accesos simultáneos.

---

9. Ejemplo completo

```
Imports System.IO

Module Module1
    Sub Main()
        Dim ruta As String = "C:\ejemplo.txt"

        ' Escribir
        File.WriteAllText(ruta, "Primera línea")

        ' Leer
        Dim texto As String = File.ReadAllText(ruta)
        Console.WriteLine("Contenido: " & texto)

        ' Agregar
        File.AppendAllText(ruta, vbCrLf & "Segunda línea")

        ' Leer línea por línea
        Dim lineas = File.ReadAllLines(ruta)
        For Each linea In lineas
            Console.WriteLine(">> " & linea)
        Next
    End Sub
End Module
```

---

14. Conexión y manejo de bases de datos con Visual Basic .NET

---

1. Introducción

Visual Basic .NET permite conectarse y manipular bases de datos utilizando ADO.NET, una colección de clases que forman parte de .NET Framework para acceder a fuentes de datos como SQL Server, MySQL, SQLite, entre otros.

---

2. Requisitos previos

- Tener instalado Visual Studio (cualquier edición con soporte para VB.NET).
- Tener acceso a una base de datos (por ejemplo, SQL Server Express).
- Conocimiento básico de SQL (SELECT, INSERT, UPDATE, DELETE).

---

3. Componentes principales de ADO.NET

- **SqlConnection**: establece la conexión con la base de datos.
- **SqlCommand**: ejecuta comandos SQL.
- **SqlDataReader**: lee los datos devueltos por una consulta.
- **SqlDataAdapter** y **DataSet**: para trabajar con datos desconectados.

---

4. Ejemplo de conexión a una base de datos SQL Server

```
Imports System.Data.SqlClient

Module Module1
    Sub Main()
        Dim connectionString As String = "Data Source=.\SQLEXPRESS;Initial
Catalog=MiBaseDeDatos;Integrated Security=True"
        Using connection As New SqlConnection(connectionString)
            Try
                connection.Open()
                Console.WriteLine("Conexión exitosa.")
            Catch ex As Exception
                Console.WriteLine("Error al conectar: " & ex.Message)
            End Try
        End Using
    End Sub
End Module
```

---

5. Ejemplo: Leer datos de una tabla

```
Dim query As String = "SELECT Nombre, Edad FROM Personas"
Dim command As New SqlCommand(query, connection)
Dim reader As SqlDataReader = command.ExecuteReader()

While reader.Read()
    Console.WriteLine("Nombre: " & reader("Nombre") & ", Edad: " &
reader("Edad"))
End While
reader.Close()
```

---

6. Insertar datos

```
Dim insertQuery As String = "INSERT INTO Personas (Nombre, Edad) VALUES
(@nombre, @edad)"
Dim insertCommand As New SqlCommand(insertQuery, connection)
insertCommand.Parameters.AddWithValue("@nombre", "Ana")
insertCommand.Parameters.AddWithValue("@edad", 30)
insertCommand.ExecuteNonQuery()
```

---

7. Actualizar datos

```
Dim updateQuery As String = "UPDATE Personas SET Edad = @edad WHERE Nombre =
@nombre"
Dim updateCommand As New SqlCommand(updateQuery, connection)
updateCommand.Parameters.AddWithValue("@nombre", "Ana")
updateCommand.Parameters.AddWithValue("@edad", 31)
updateCommand.ExecuteNonQuery()
```

---

8. Eliminar datos

```
Dim deleteQuery As String = "DELETE FROM Personas WHERE Nombre = @nombre"
Dim deleteCommand As New SqlCommand(deleteQuery, connection)
deleteCommand.Parameters.AddWithValue("@nombre", "Ana")
deleteCommand.ExecuteNonQuery()
```

---

9. Uso de DataSet y DataAdapter (modo desconectado)

```
Dim adapter As New SqlDataAdapter("SELECT * FROM Personas", connection)
Dim ds As New DataSet()
adapter.Fill(ds, "Personas")

For Each row As DataRow In ds.Tables("Personas").Rows
    Console.WriteLine(row("Nombre") & " - " & row("Edad"))
Next
```

---

10. Conclusión

Con ADO.NET en Visual Basic .NET se puede crear aplicaciones robustas para interactuar con bases de datos. Se recomienda implementar capas separadas para manejo de datos (DAL) en proyectos grandes, así como proteger las consultas con parámetros para evitar inyecciones SQL.

---

15. Creación de Interfaces Gráficas (Windows Forms) en Visual Basic .NET

1. Introducción a Windows Forms

Windows Forms es una biblioteca de clases dentro de .NET Framework utilizada para crear aplicaciones de escritorio con interfaz gráfica en sistemas Windows. Permite diseñar formularios (ventanas) con controles visuales como botones, etiquetas, cuadros de texto, etc.

2. Requisitos Previos

- Tener instalado Visual Studio (versión Community o superior)
- Conocimiento básico de Visual Basic .NET
- Familiaridad con el entorno gráfico de Windows

---

3. Estructura Básica de un Formulario

Cuando se crea un nuevo proyecto de tipo “Windows Forms App” en Visual Basic, se genera automáticamente un formulario principal (`Form1.vb`). Este formulario se puede diseñar visualmente desde el diseñador o mediante código.

Ejemplo básico de código:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.Text = "Mi Primer Formulario"
    End Sub
End Class
```

---

4. Controles Comunes y su Uso

Control	Descripción	Propiedades Clave
Label	Muestra texto estático	`Text`, `Font`, `ForeColor`
TextBox	Entrada de texto	`Text`, `Multiline`, `ReadOnly`
Button	Botón interactivo	`Text`, `Click (evento)`
CheckBox	Casilla de verificación	`Checked`, `Text`
ComboBox	Lista desplegable	`Items`, `SelectedItem`
ListBox	Lista de selección	`Items`, `SelectedIndex`
PictureBox	Imagen en el formulario	`Image`, `SizeMode`

---

5. Manejo de Eventos

Cada control puede generar eventos, como `Click`, `TextChanged`, `CheckedChanged`, etc. Se usan para ejecutar código cuando el usuario interactúa con la interfaz.

Ejemplo: Evento Click de un botón

```
Private Sub btnSaludar_Click(sender As Object, e As EventArgs) Handles
btnSaludar.Click
    MessageBox.Show("Hola, " & txtNombre.Text)
End Sub
```

---

6. Diseño del Formulario

El diseñador de Visual Studio permite arrastrar y soltar controles. Desde el panel de propiedades se pueden cambiar atributos como el tamaño, color, nombre del control, etc.

Buenas prácticas:

- Cambiar el nombre de los controles (`btnAceptar`, `lblMensaje`) en lugar de usar los nombres por defecto.
- Organizar visualmente los controles usando `GroupBox`, `Panel` o `TableLayoutPanel`.

---

7. Validación de Datos

Es común validar entradas antes de procesarlas.

```
If txtEdad.Text = "" Or Not IsNumeric(txtEdad.Text) Then
    MessageBox.Show("Por favor, ingresa una edad válida.")
    Exit Sub
End If
```

---

8. Ventanas Múltiples

Puedes abrir otros formularios desde el principal:

```
Dim frm As New Form2()
frm.Show()
```

---

9. Guardar y Cargar Información

Es posible leer y escribir datos en archivos:

Guardar en archivo de texto:

```
IO.File.WriteAllText("datos.txt", txtDatos.Text)
```

Leer desde archivo de texto:

```
txtDatos.Text = IO.File.ReadAllText("datos.txt")
```

---

10. Compilar y Ejecutar la Aplicación

Presiona **F5** o haz clic en “Iniciar” para compilar y ejecutar tu aplicación. Asegúrate de que no haya errores de sintaxis ni de lógica.

---

11. Ejercicio Práctico

Objetivo: Crear una aplicación que solicite nombre y edad, y muestre un saludo personalizado.

Controles necesarios:

- 2 Label (`lblNombre`, `lblEdad`)
- 2 TextBox (`txtNombre`, `txtEdad`)
- 1 Button (`btnSaludar`)
- 1 Label (`lblResultado`)

Código del botón:

```
Private Sub btnSaludar_Click(sender As Object, e As EventArgs) Handles
btnSaludar.Click
    Dim nombre As String = txtNombre.Text
    Dim edad As String = txtEdad.Text

    If nombre = "" Or edad = "" Or Not IsNumeric(edad) Then
        MessageBox.Show("Por favor, completa todos los campos correctamente.")
        Exit Sub
    End If

    lblResultado.Text = "Hola " & nombre & ", tienes " & edad & " años."
End Sub
```

---

12. Conclusión

Windows Forms es una forma sencilla y potente de desarrollar interfaces gráficas en Visual Basic .NET. Ideal para aplicaciones de escritorio pequeñas y medianas, permite un desarrollo rápido y visual.

---

16. Aplicaciones Multinivel y Modulares en Visual Basic .NET

1. Introducción

Visual Basic .NET (VB.NET) permite desarrollar aplicaciones robustas y escalables, gracias a su soporte para la programación modular y su arquitectura flexible para aplicaciones multinivel. Estas características son esenciales en el desarrollo profesional de software empresarial.

---

2. ¿Qué es una Aplicación Multinivel?

Una aplicación multinivel (o **n-tier**) es aquella que separa su lógica en capas bien definidas. Las capas principales son:

- **Capa de presentación (UI)**: Interfaz que interactúa con el usuario.
- **Capa de lógica de negocio (BLL)**: Procesa las reglas del negocio.
- **Capa de acceso a datos (DAL)**: Gestiona la comunicación con la base de datos.

Ventajas:

- Mantenibilidad y escalabilidad.
- Separación de responsabilidades.
- Reutilización de código.

---

3. ¿Qué es una Aplicación Modular?

Una aplicación modular está compuesta por partes independientes (módulos) que se pueden desarrollar, mantener y probar por separado.

Beneficios:

- Código más limpio y organizado.
- Facilita el trabajo en equipo.
- Mejora la capacidad de prueba y reutilización.

---

4. Estructura de una Aplicación Multinivel Modular en VB.NET

Una forma típica de organizar una aplicación en VB.NET es usando **proyectos independientes** dentro de una **solución**:

- **Proyecto UI (Windows Forms o WPF)**
- **Proyecto de Biblioteca de Clases para la BLL**
- **Proyecto de Biblioteca de Clases para la DAL**
- **Proyecto de Entidades (opcional)**

---

5. Ejemplo Básico

Supongamos una aplicación para gestionar productos.

5.1. Capa de Entidades (`Entidades`)

```
Public Class Producto
    Public Property Id As Integer
    Public Property Nombre As String
    Public Property Precio As Decimal
End Class
```

5.2. Capa de Acceso a Datos (`DAL`)

```
Public Class ProductoDAL
    Public Shared Function ObtenerProductos() As List(Of Producto)
        ' Simula la lectura de base de datos
        Return New List(Of Producto) From {
            New Producto With {.Id = 1, .Nombre = "Teclado", .Precio = 25},
            New Producto With {.Id = 2, .Nombre = "Ratón", .Precio = 15}
        }
    End Function
End Class
```

5.3. Capa de Lógica de Negocio (`BLL`)

```
Public Class ProductoBLL
    Public Shared Function ListarProductos() As List(Of Producto)
        Return ProductoDAL.ObtenerProductos()
    End Function
End Class
```

5.4. Capa de Presentación (`UI`)

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim productos = ProductoBLL.ListarProductos()
    DataGridView1.DataSource = productos
End Sub
```

---

6. Buenas Prácticas

- Utilizar interfaces para facilitar el desacoplamiento.
- Separar proyectos por funcionalidad.
- Aplicar principios SOLID.
- Usar inyección de dependencias para mayor flexibilidad.

---

7. Herramientas Sugeridas

- Visual Studio
- SQL Server o SQLite
- Entity Framework (opcional)

---

8. Conclusión

El desarrollo de aplicaciones multinivel y modulares en VB.NET mejora la organización del código, facilita el mantenimiento y permite escalar las aplicaciones de forma eficiente. Adoptar esta arquitectura es esencial para proyectos medianos y grandes en entornos profesionales.

---

17. Integración de Librerías Externas en Visual Basic .NET

1. ¿Qué es una librería externa?

Una **librería externa** es un conjunto de funciones, clases o recursos predefinidos que podemos reutilizar en nuestros proyectos sin tener que escribir todo desde cero. En Visual Basic .NET (VB.NET), las librerías pueden ser:

- **DLLs (Dynamic Link Libraries)**: Escritas en .NET o en lenguajes no administrados (como C/C++).
- **Librerías de terceros en .NET**: Como `Newtonsoft.Json`, `MySql.Data`, etc.

---

2. Tipos de librerías en VB.NET

Tipo de librería	Ejemplo	Cómo se usa
.NET (DLL)	`System.IO.dll`	Referencia directa en el proyecto
Terceros (NuGet)	`Newtonsoft.Json`	Instalación vía NuGet
COM	`Microsoft Excel Object Library`	Agregar como referencia COM
No administrada	`kernel32.dll` (API de Windows)	Invocación con `Declare` o `DllImport`

---

3. Cómo agregar una librería .DLL en VB.NET

Paso 1: Abrir el proyecto en Visual Studio

1. Abre tu proyecto.
2. En el **Explorador de soluciones**, haz clic derecho sobre “Referencias”.
3. Selecciona “Agregar referencia…”.

Paso 2: Buscar la DLL

1. En el cuadro de diálogo, ve a la pestaña “Examinar”.
2. Busca la ruta de la DLL externa.
3. Haz clic en “Aceptar”.

Paso 3: Importar el espacio de nombres

En tu archivo `.vb`, escribe:

```
Imports NombreDeLaLibreria
```

---

4. Cómo instalar una librería desde NuGet

Paso 1: Abrir la consola de NuGet

Ve a:

```
Herramientas > Administrador de paquetes NuGet > Consola del Administrador de paquetes
```

Paso 2: Instalar el paquete

Escribe, por ejemplo:

```
Install-Package Newtonsoft.Json
```

Paso 3: Usar la librería en tu código

```
Imports Newtonsoft.Json

Dim jsonString As String = JsonConvert.SerializeObject(New With {.Nombre = "Ana", .Edad = 30})
```

---

5. Usar librerías no administradas (DLLs nativas)

Ejemplo: Llamar a funciones de Windows API

```
Imports System.Runtime.InteropServices

Public Class ApiExample
    <DllImport("user32.dll", CharSet:=CharSet.Auto)> _
    Public Shared Function MessageBox(ByVal hWnd As IntPtr, ByVal text As String, _
                                      ByVal caption As String, ByVal type As UInteger) As Integer
    End Function

    Public Sub MostrarMensaje()
        MessageBox(IntPtr.Zero, "Hola desde la API de Windows", "Mensaje", 0)
    End Sub
End Class
```

---

6. Buenas prácticas

- **Verifica compatibilidad .NET Framework/Core** de la librería.
- **Lee la documentación del proveedor** si es de terceros.
- **Agrupa las librerías** en una carpeta si vas a redistribuir tu software.
- **Firma las DLL** si trabajas en un entorno corporativo con seguridad fuerte.

7. Errores comunes

Error	Causa probable
“No se puede cargar la DLL…”	Ruta incorrecta, versión incompatible
“El tipo no se define…”	Faltó importar el espacio de nombres
Conflictos de versión	Múltiples versiones de una misma librería
Error en tiempo de ejecución	DLL no encontrada al ejecutar (faltó copiarla)

---

8. Conclusión

Integrar librerías externas en VB.NET permite ampliar el poder de tus aplicaciones, ya sea reutilizando código propio, utilizando librerías de terceros o integrando servicios externos. Con Visual Studio y NuGet, este proceso es cada vez más sencillo y robusto.

---

18. Proyecto de ejemplo paso a paso

Ejercicio 1: Mostrar un mensaje por consola

Enunciado:
Crea un programa que muestre el mensaje "Hola, mundo" en la consola.

Explicación:
Este ejercicio es básico y sirve para familiarizarse con la estructura de un programa en VB.NET y con la consola como salida. Se utiliza el método `Console.WriteLine` para imprimir texto.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Hola, mundo")
        Console.ReadLine() ' Para que la consola no se cierre inmediatamente
    End Sub
End Module
```

Ejercicio 2: Sumar dos números ingresados por el usuario

Enunciado:
Escribe un programa que pida al usuario dos números y muestre su suma.

Explicación:
Este ejercicio introduce la lectura de datos desde la consola con `Console.ReadLine()`, la conversión de cadenas a números (`Convert.ToInt32`), y la suma simple.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa el primer número:")
        Dim num1 As Integer = Convert.ToInt32(Console.ReadLine())

        Console.WriteLine("Ingresa el segundo número:")
        Dim num2 As Integer = Convert.ToInt32(Console.ReadLine())

        Dim suma As Integer = num1 + num2
        Console.WriteLine("La suma es: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 3: Determinar si un número es par o impar

Enunciado:
Solicita al usuario un número y determina si es par o impar.

Explicación:
Aquí se introduce el uso de la estructura condicional `If...Else` y el operador módulo `%` para obtener el residuo de una división.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa un número:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())

        If num Mod 2 = 0 Then
            Console.WriteLine("El número es par.")
        Else
            Console.WriteLine("El número es impar.")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Genial, entonces prepararé aquí un documento con los 100 ejercicios completos de Visual Basic .NET, progresivos en dificultad, con explicación y solución.

Lo haré en bloques para que no sea demasiado largo de leer de golpe. Aquí tienes los primeros 20 ejercicios:

---

100 Ejercicios Progresivos en Visual Basic .NET

Con explicación y solución

Ejercicio 1: Mostrar un mensaje por consola

Enunciado: Mostrar "Hola, mundo" en consola.
Explicación: Uso básico de `Console.WriteLine`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Hola, mundo")
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 2: Sumar dos números ingresados

Enunciado: Pedir dos números y mostrar su suma.
Explicación: Leer con `Console.ReadLine()`, convertir a Integer y sumar.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa primer número:")
        Dim n1 As Integer = Convert.ToInt32(Console.ReadLine())
        Console.WriteLine("Ingresa segundo número:")
        Dim n2 As Integer = Convert.ToInt32(Console.ReadLine())
        Console.WriteLine("Suma: " & (n1 + n2))
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 3: Par o impar

Enunciado: Dado un número, indicar si es par o impar.
Explicación: Uso de `If` y operador módulo `%`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa un número:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())
        If num Mod 2 = 0 Then
            Console.WriteLine("Es par")
        Else
            Console.WriteLine("Es impar")
        End If
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 4: Número mayor

Enunciado: Pedir dos números y mostrar cuál es mayor.
Explicación: Uso de condicional `If...ElseIf`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número 1:")
        Dim a As Integer = Convert.ToInt32(Console.ReadLine())
        Console.WriteLine("Número 2:")
        Dim b As Integer = Convert.ToInt32(Console.ReadLine())

        If a > b Then
            Console.WriteLine("Mayor: " & a)
        ElseIf b > a Then
            Console.WriteLine("Mayor: " & b)
        Else
            Console.WriteLine("Son iguales")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 5: Tabla de multiplicar

Enunciado: Mostrar la tabla de multiplicar de un número ingresado.
Explicación: Uso de bucle `For`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número para tabla:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())

        For i As Integer = 1 To 10
            Console.WriteLine(num & " x " & i & " = " & (num * i))
        Next

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 6: Sumar números hasta cero

Enunciado: Pedir números al usuario y sumar hasta que ingrese cero.
Explicación: Bucle `While` con condición de salida.
Solución:

```
Module Module1
    Sub Main()
        Dim suma As Integer = 0
        Dim num As Integer

        Do
            Console.WriteLine("Ingresa un número (0 para salir):")
            num = Convert.ToInt32(Console.ReadLine())
            suma += num
        Loop While num <> 0

        Console.WriteLine("Suma total: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 7: Contar números positivos

Enunciado: Pedir 10 números y mostrar cuántos son positivos.
Explicación: Uso de contador y ciclo `For`.
Solución:

```
Module Module1
    Sub Main()
        Dim countPos As Integer = 0

        For i As Integer = 1 To 10
            Console.WriteLine("Número " & i & ":")
            Dim num As Integer = Convert.ToInt32(Console.ReadLine())
            If num > 0 Then
                countPos += 1
            End If
        Next

        Console.WriteLine("Números positivos: " & countPos)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 8: Factorial de un número

Enunciado: Calcular el factorial de un número entero.
Explicación: Uso de bucle `For` para multiplicar sucesivamente.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número para factorial:")
        Dim n As Integer = Convert.ToInt32(Console.ReadLine())
        Dim factorial As Integer = 1

        For i As Integer = 1 To n
            factorial *= i
        Next

        Console.WriteLine("Factorial de " & n & " es " & factorial)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 9: Invertir un número

Enunciado: Pedir un número y mostrarlo invertido (123 -> 321).
Explicación: Uso de manipulación de cadenas o matemática con módulo y división.
Solución:

```
Module Module1

    Sub Main()

        Console.WriteLine("Número a invertir:")

        Dim num As Integer = Convert.ToInt32(Console.ReadLine())

        Dim invertido As Integer = 0


        While num > 0

            invertido = invertido * 10 + (num Mod 10)

            num \= 10

        End While


        Console.WriteLine("Número invertido: " & invertido)

        Console.ReadLine()

    End Sub

End Module
```

Ejercicio 10: Números primos hasta N

Enunciado: Mostrar todos los números primos hasta un número N ingresado.
Explicación: Uso de ciclos anidados para verificar divisibilidad.
Solución:

```
Module Module1
    Function EsPrimo(num As Integer) As Boolean
        If num < 2 Then Return False
        For i As Integer = 2 To Math.Sqrt(num)
            If num Mod i = 0 Then Return False
        Next
        Return True
    End Function

    Sub Main()
        Console.WriteLine("Mostrar primos hasta:")
        Dim N As Integer = Convert.ToInt32(Console.ReadLine())

        For i As Integer = 2 To N
            If EsPrimo(i) Then
                Console.WriteLine(i)
            End If
        Next

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 11: Promedio de números

Enunciado: Pedir N números y mostrar el promedio.
Explicación: Sumar y contar, luego dividir.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("¿Cuántos números ingresarás?")
        Dim N As Integer = Convert.ToInt32(Console.ReadLine())
        Dim suma As Double = 0

        For i As Integer = 1 To N
            Console.WriteLine("Número " & i & ":")
            suma += Convert.ToDouble(Console.ReadLine())
        Next

        Console.WriteLine("Promedio: " & (suma / N))
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 12: Número Fibonacci en posición N

Enunciado: Mostrar el número Fibonacci en la posición N.
Explicación: Serie Fibonacci: cada número es suma de los dos anteriores.
Solución:

```
Module Module1
    Function Fibonacci(n As Integer) As Integer
        If n <= 1 Then
            Return n
        Else
            Return Fibonacci(n - 1) + Fibonacci(n - 2)
        End If
    End Function

    Sub Main()
        Console.WriteLine("Posición en Fibonacci:")
        Dim n As Integer = Convert.ToInt32(Console.ReadLine())
        Console.WriteLine("Número Fibonacci: " & Fibonacci(n))
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 13: Palíndromo

Enunciado: Pedir una palabra y determinar si es palíndromo (se lee igual al derecho y al revés).
Explicación: Comparar la palabra con su inversa.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa una palabra:")
        Dim palabra As String = Console.ReadLine()
        Dim invertida As String = ""

        For i As Integer = palabra.Length - 1 To 0 Step -1
            invertida &= palabra(i)
        Next

        If palabra.ToLower() = invertida.ToLower() Then
            Console.WriteLine("Es palíndromo")
        Else
            Console.WriteLine("No es palíndromo")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 14: Contar vocales en una cadena

Enunciado: Pedir una cadena y contar cuántas vocales tiene.
Explicación: Recorrer cadena y verificar si cada letra es vocal.
Solución:

```
Module Module1

    Sub Main()

        Console.WriteLine("Ingresa una frase:")

        Dim frase As String = Console.ReadLine().ToLower()

        Dim count As Integer = 0

        Dim vocales As String = "aeiou"


        For Each c As Char In frase

            If vocales.Contains(c) Then

                count += 1

            End If

        Next


        Console.WriteLine("Número de vocales: " & count)

        Console.ReadLine()

    End Sub

End Module
```

Ejercicio 15: Contar palabras en una frase

Enunciado: Pedir una frase y mostrar cuántas palabras tiene.
Explicación: Usar método `Split` para separar palabras.
Solución:

```
Module Module1

    Sub Main()

        Console.WriteLine("Ingresa una frase:")

        Dim frase As String = Console.ReadLine()

        Dim palabras() As String = frase.Split(New Char() {" "c},
StringSplitOptions.RemoveEmptyEntries)

        Console.WriteLine("Número de palabras: " & palabras.Length)

        Console.ReadLine()

    End Sub

End Module
```

Ejercicio 16: Ordenar un arreglo de números

Enunciado: Ordenar un arreglo de números ingresados por el usuario.
Explicación: Uso del método `Array.Sort`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("¿Cuántos números deseas ingresar?")
        Dim n As Integer = Convert.ToInt32(Console.ReadLine())
        Dim nums(n - 1) As Integer

        For i As Integer = 0 To n - 1
            Console.WriteLine("Número " & (i + 1) & ":")
            nums(i) = Convert.ToInt32(Console.ReadLine())
        Next

        Array.Sort(nums)

        Console.WriteLine("Números ordenados:")
        For Each num As Integer In nums
            Console.WriteLine(num)
        Next

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 17: Buscar un número en un arreglo

Enunciado: Pedir un arreglo y un número, y decir si está en el arreglo.
Explicación: Uso de `Array.IndexOf`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Cantidad de números:")
        Dim n As Integer = Convert.ToInt32(Console.ReadLine())
        Dim arr(n - 1) As Integer

        For i As Integer = 0 To n - 1
            Console.WriteLine("Número " & (i + 1) & ":")
            arr(i) = Convert.ToInt32(Console.ReadLine())
        Next

        Console.WriteLine("Número a buscar:")
        Dim buscar As Integer = Convert.ToInt32(Console.ReadLine())

        If Array.IndexOf(arr, buscar) >= 0 Then
            Console.WriteLine("Número encontrado")
        Else
            Console.WriteLine("Número no encontrado")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 18: Sumar elementos pares de un arreglo

Enunciado: Sumar sólo los números pares de un arreglo.
Explicación: Recorrer arreglo y sumar sólo los pares.
Solución:

```
Module Module1

    Sub Main()

        Console.WriteLine("Cantidad de números:")

        Dim n As Integer = Convert.ToInt32(Console.ReadLine())

        Dim arr(n - 1) As Integer

        Dim suma As Integer = 0


        For i As Integer = 0 To n - 1

            Console.WriteLine("Número " & (i + 1) & ":")

            arr(i) = Convert.ToInt32(Console.ReadLine())

            If arr(i) Mod 2 = 0 Then

                suma += arr(i)

            End If

        Next


        Console.WriteLine("Suma de pares: " & suma)

        Console.ReadLine()

    End Sub

End Module
```

Ejercicio 19: Número más pequeño en un arreglo

Enunciado: Encontrar el número mínimo de un arreglo.
Explicación: Recorrer arreglo para comparar.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Cantidad de números:")
        Dim n As Integer = Convert.ToInt32(Console.ReadLine())
        Dim arr(n - 1) As Integer

        For i As Integer = 0 To n - 1
            Console.WriteLine("Número " & (i + 1) & ":")
            arr(i) = Convert.ToInt32(Console.ReadLine())
        Next

        Dim min As Integer = arr(0)
        For Each num As Integer In arr
            If num < min Then
                min = num
            End If
        Next

        Console.WriteLine("Número mínimo: " & min)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 20: Conversión de grados Celsius a Fahrenheit

Enunciado: Convertir grados Celsius a Fahrenheit.
Explicación: Aplicar fórmula F = C * 9/5 + 32.
Solución:

```
Module Module1

    Sub Main()

        Console.WriteLine("Grados Celsius:")

        Dim celsius As Double = Convert.ToDouble(Console.ReadLine())

        Dim fahrenheit As Double = celsius * 9 / 5 + 32

        Console.WriteLine("Grados Fahrenheit: " & fahrenheit)

        Console.ReadLine()

    End Sub

End Module
```

Ejercicio 21: Calcular área de un círculo

Enunciado: Pedir el radio y calcular el área del círculo.
Explicación: Área = π * radio², usar `Math.PI` y `Math.Pow`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Radio del círculo:")
        Dim radio As Double = Convert.ToDouble(Console.ReadLine())
        Dim area As Double = Math.PI * Math.Pow(radio, 2)
        Console.WriteLine("Área del círculo: " & area)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 22: Número invertido como cadena

Enunciado: Pedir un número y mostrarlo invertido usando cadenas.
Explicación: Convertir a cadena, usar `StrReverse`.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número a invertir:")
        Dim num As String = Console.ReadLine()
        Dim invertido As String = StrReverse(num)
        Console.WriteLine("Número invertido: " & invertido)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 23: Generar un arreglo con números aleatorios

Enunciado: Crear un arreglo de 10 números aleatorios entre 1 y 100.
Explicación: Uso de clase `Random`.
Solución:

```
Module Module1
    Sub Main()
        Dim rnd As New Random()
        Dim arr(9) As Integer

        For i As Integer = 0 To 9
            arr(i) = rnd.Next(1, 101)
        Next

        Console.WriteLine("Arreglo generado:")
        For Each num As Integer In arr
            Console.WriteLine(num)
        Next

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 24: Sumar elementos de un arreglo

Enunciado: Sumar todos los elementos de un arreglo dado.
Explicación: Recorrer arreglo y acumular suma.
Solución:

```
Module Module1
    Sub Main()
        Dim arr() As Integer = {2, 4, 6, 8, 10}
        Dim suma As Integer = 0

        For Each num As Integer In arr
            suma += num
        Next

        Console.WriteLine("Suma de elementos: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 25: Convertir decimal a binario

Enunciado: Pedir un número decimal y mostrar su equivalente en binario.
Explicación: Uso de división repetida por 2 y concatenación.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número decimal:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())
        Dim binario As String = ""

        While num > 0
            binario = (num Mod 2).ToString() & binario
            num \= 2
        End While

        Console.WriteLine("Binario: " & binario)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 26: Encontrar máximo y mínimo en arreglo

Enunciado: Dado un arreglo, encontrar el valor máximo y mínimo.
Explicación: Recorrer arreglo y comparar para actualizar máximo y mínimo.
Solución:

```
Module Module1
    Sub Main()
        Dim arr() As Integer = {5, 9, 1, 7, 3}
        Dim max As Integer = arr(0)
        Dim min As Integer = arr(0)

        For Each num As Integer In arr
            If num > max Then max = num
            If num < min Then min = num
        Next

        Console.WriteLine("Máximo: " & max)
        Console.WriteLine("Mínimo: " & min)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 27: Convertir número a texto (1 a 5)

Enunciado: Pedir un número entre 1 y 5 y mostrarlo en texto (1 -> "Uno").
Explicación: Uso de Select Case.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número entre 1 y 5:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())
        Select Case num
            Case 1
                Console.WriteLine("Uno")
            Case 2
                Console.WriteLine("Dos")
            Case 3
                Console.WriteLine("Tres")
            Case 4
                Console.WriteLine("Cuatro")
            Case 5
                Console.WriteLine("Cinco")
            Case Else
                Console.WriteLine("Número fuera de rango")
        End Select
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 28: Comprobar año bisiesto

Enunciado: Pedir un año y determinar si es bisiesto.
Explicación: Año divisible por 4 y no por 100, o divisible por 400.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Año:")
        Dim año As Integer = Convert.ToInt32(Console.ReadLine())
        Dim bisiesto As Boolean = False

        If (año Mod 4 = 0 And año Mod 100 <> 0) Or (año Mod 400 = 0) Then
            bisiesto = True
        End If

        If bisiesto Then
            Console.WriteLine(año & " es bisiesto")
        Else
            Console.WriteLine(año & " no es bisiesto")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 29: Suma de dígitos

Enunciado: Pedir un número y sumar sus dígitos.
Explicación: Convertir a cadena o usar módulo y división.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())
        Dim suma As Integer = 0

        While num > 0
            suma += num Mod 10
            num \= 10
        End While

        Console.WriteLine("Suma de dígitos: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 30: Crear y mostrar matriz 3x3

Enunciado: Crear una matriz 3x3 con números ingresados y mostrarla.
Explicación: Uso de arreglos bidimensionales.
Solución:

```
Module Module1
    Sub Main()
        Dim matriz(2, 2) As Integer

        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                Console.WriteLine($"Ingresa número para posición ({i},{j}):")
                matriz(i, j) = Convert.ToInt32(Console.ReadLine())
            Next
        Next

        Console.WriteLine("Matriz ingresada:")
        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                Console.Write(matriz(i, j) & " ")
            Next
            Console.WriteLine()
        Next

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 31: Sumar diagonal principal de matriz

Enunciado: Sumar los elementos de la diagonal principal de una matriz cuadrada.

Explicación: Sumar elementos donde fila = columna.

Solución:

```
Module Module1
    Sub Main()
        Dim matriz(2, 2) As Integer
        Dim suma As Integer = 0

        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                Console.WriteLine($"Número para ({i},{j}):")
                matriz(i, j) = Convert.ToInt32(Console.ReadLine())
            Next
        Next

        For i As Integer = 0 To 2
            suma += matriz(i, i)
        Next

        Console.WriteLine("Suma diagonal principal: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 32: Contar caracteres en una cadena

Enunciado: Pedir una cadena y mostrar su longitud.

Explicación: Uso de propiedad `.Length`.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa una cadena:")
        Dim cadena As String = Console.ReadLine()
        Console.WriteLine("Longitud: " & cadena.Length)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 33: Contar caracteres específicos

Enunciado: Contar cuántas veces aparece una letra específica en una cadena.

Explicación: Recorrer cadena y comparar caracteres.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Cadena:")
        Dim cadena As String = Console.ReadLine()
        Console.WriteLine("Letra a contar:")
        Dim letra As Char = Convert.ToChar(Console.ReadLine())
        Dim count As Integer = 0

        For Each c As Char In cadena
            If c = letra Then
                count += 1
            End If
        Next

        Console.WriteLine($"La letra '{letra}' aparece {count} veces.")
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 34: Convertir texto a mayúsculas

Enunciado: Pedir texto y mostrarlo en mayúsculas.

Explicación: Uso de método `.ToUpper()`.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Texto:")
        Dim texto As String = Console.ReadLine()
        Console.WriteLine("Mayúsculas: " & texto.ToUpper())
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 35: Convertir texto a minúsculas

Enunciado: Pedir texto y mostrarlo en minúsculas.

Explicación: Uso de método `.ToLower()`.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Texto:")
        Dim texto As String = Console.ReadLine()
        Console.WriteLine("Minúsculas: " & texto.ToLower())
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 36: Sumar matriz 3x3

Enunciado: Sumar todos los elementos de una matriz 3x3.

Explicación: Recorrer todos los elementos y acumular.

Solución:

```
Module Module1
    Sub Main()
        Dim matriz(2, 2) As Integer
        Dim suma As Integer = 0

        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                Console.WriteLine($"Número para ({i},{j}):")
                matriz(i, j) = Convert.ToInt32(Console.ReadLine())
            Next
        Next

        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                suma += matriz(i, j)
            Next
        Next

        Console.WriteLine("Suma total de matriz: " & suma)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 37: Mostrar letras con índice

Enunciado: Mostrar cada letra de una palabra con su índice.

Explicación: Recorrer cadena con índice.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Palabra:")
        Dim palabra As String = Console.ReadLine()

        For i As Integer = 0 To palabra.Length - 1
            Console.WriteLine($"Índice {i}: {palabra(i)}")
        Next

        Console.ReadLine()
    End Sub
End Module
``
```

Ejercicio 38: Validar si una cadena es un palíndromo

Enunciado: Pedir una cadena y determinar si es palíndromo (se lee igual al derecho y al revés).
Explicación: Convertir a minúsculas, eliminar espacios, comparar con la cadena invertida.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa una cadena:")
        Dim texto As String = Console.ReadLine().ToLower().Replace(" ", "")
        Dim invertido As String = StrReverse(texto)

        If texto = invertido Then
            Console.WriteLine("Es palíndromo")
        Else
            Console.WriteLine("No es palíndromo")
        End If

        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 39: Contar vocales en una cadena

Enunciado: Pedir una cadena y contar cuántas vocales contiene.
Explicación: Recorrer cadena y sumar si el carácter es vocal.
Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Ingresa una cadena:")
        Dim texto As String = Console.ReadLine().ToLower()
        Dim vocales As String = "aeiou"
        Dim contador As Integer = 0

        For Each c As Char In texto
            If vocales.Contains(c) Then
                contador += 1
            End If
        Next

        Console.WriteLine("Número de vocales: " & contador)
        Console.ReadLine()
    End Sub
End Module
```

Ejercicio 40: Calcular factorial de un número

Enunciado: Pedir un número y calcular su factorial.

Explicación: Multiplicar todos los números desde 1 hasta el número dado.

Solución:

```
Module Module1
    Sub Main()
        Console.WriteLine("Número para factorial:")
        Dim num As Integer = Convert.ToInt32(Console.ReadLine())
        Dim factorial As Long = 1

        For i As Integer = 1 To num
            factorial *= i
        Next

        Console.WriteLine($"Factorial de {num} es {factorial}")
        Console.ReadLine()
    End Sub
End Module
```

---

19. Manejo de Fechas en Visual Basic .NET

---

1. Introducción

En Visual Basic .NET, el tipo de dato para trabajar con fechas y horas es `Date`. La clase `DateTime` del .NET Framework proporciona métodos y propiedades para crear, manipular y dar formato a fechas y horas.

---

2. Declaración y Asignación de Fechas

```
Dim fechaActual As Date = Date.Now
Dim fechaEspecifica As Date = #06/26/2025#
Dim otraFecha As Date = New Date(2025, 6, 26, 14, 30, 0)
```

- `Date.Now`: devuelve la fecha y hora actual.
- `#MM/dd/yyyy#`: forma literal para representar fechas.
- `New Date(año, mes, día, hora, minuto, segundo)`: constructor detallado.

---

3. Formatos de Fecha

```
Dim fecha As Date = Date.Now
Console.WriteLine(fecha.ToShortDateString())   ' 26/06/2025
Console.WriteLine(fecha.ToLongDateString())    ' jueves, 26 de junio de 2025
Console.WriteLine(fecha.ToString("dd-MM-yyyy")) ' 26-06-2025
```

Puedes personalizar los formatos usando patrones como:

- `"dd/MM/yyyy"` → día/mes/año
- `"yyyy-MM-dd"` → formato ISO
- `"dddd"` → día de la semana

---

4. Operaciones con Fechas

```
Dim hoy As Date = Date.Today
Dim mañana As Date = hoy.AddDays(1)
Dim laProximaSemana As Date = hoy.AddDays(7)
Dim haceUnMes As Date = hoy.AddMonths(-1)
```

- `AddDays(n)`, `AddMonths(n)`, `AddYears(n)` para sumar o restar tiempo.
- `DateDiff` permite calcular diferencias:

```
Dim dias As Long = DateDiff(DateInterval.Day, #06/01/2025#, #06/26/2025#)
Console.WriteLine("Días de diferencia: " & dias)
```

---

5. Comparación de Fechas

```
If fecha1 > fecha2 Then
    Console.WriteLine("La primera fecha es posterior")
ElseIf fecha1 < fecha2 Then
    Console.WriteLine("La primera fecha es anterior")
Else
    Console.WriteLine("Las fechas son iguales")
End If
```

La comparación de fechas es directa usando operadores relacionales (>, <, =, etc.).

---

6. Extraer Componentes de una Fecha

```
Dim fecha As Date = Date.Now
Dim dia As Integer = fecha.Day
Dim mes As Integer = fecha.Month
Dim año As Integer = fecha.Year
Dim hora As Integer = fecha.Hour
Dim minuto As Integer = fecha.Minute
```

---

7. Conversión de Cadenas a Fecha

```
Dim fechaTexto As String = "26/06/2025"
Dim fechaConvertida As Date = Date.Parse(fechaTexto)
```

Para evitar errores:

```
Dim exito As Boolean = Date.TryParse(fechaTexto, fechaConvertida)
If exito Then
    Console.WriteLine("Fecha válida: " & fechaConvertida)
Else
    Console.WriteLine("Fecha no válida")
End If
```

---

8. Fechas y Funciones de Calendario

Puedes usar funciones avanzadas con `CultureInfo` y `Calendar` si necesitas trabajar con calendarios específicos, formatos regionales, semanas del año, etc.

```
Dim cultura As New Globalization.CultureInfo("es-ES")
Dim nombreMes As String = fecha.ToString("MMMM", cultura)
```

---

9. Ejemplo Completo

```
Module Program
    Sub Main()
        Dim hoy As Date = Date.Today
        Console.WriteLine("Hoy es: " & hoy.ToLongDateString())
        Console.WriteLine("Dentro de 10 días: " &
hoy.AddDays(10).ToShortDateString())
        Dim fechaTexto As String = "15/07/2025"
        Dim fechaConvertida As Date
        If Date.TryParse(fechaTexto, fechaConvertida) Then
            Console.WriteLine("Fecha válida: " & fechaConvertida)
        Else
            Console.WriteLine("Fecha inválida")
        End If
    End Sub
End Module
```

---

10. Conclusión

Visual Basic .NET facilita el trabajo con fechas gracias al tipo `Date` y la clase `DateTime`. Saber manipular fechas es esencial para programas que gestionan eventos, agendas, vencimientos, etc.

---

20. Manejo del Color en Visual Basic .NET

---

1. Introducción

Visual Basic .NET permite trabajar con colores en interfaces gráficas a través de clases como `System.Drawing.Color` y propiedades específicas de controles. Comprender cómo manipular colores es esencial para personalizar formularios, botones, textos y otros elementos visuales.

---

2. Clase `System.Drawing.Color`

La clase `Color` proporciona colores predefinidos y permite crear colores personalizados. Algunos ejemplos:

```
Dim color1 As Color = Color.Red          ' Color rojo
Dim color2 As Color = Color.FromArgb(255, 0, 255) ' Color personalizado (fucsia)
Dim color3 As Color = Color.FromName("Blue")      ' Desde el nombre
```

---

3. Usar colores en controles

Cambiar el color de fondo de un formulario:

```
Me.BackColor = Color.LightBlue
```

Cambiar el color del texto en un `Label`:

```
Label1.ForeColor = Color.Green
```

Cambiar el color de fondo de un `Button`:

```
Button1.BackColor = Color.Orange
```

---

4. Crear colores personalizados con `FromArgb`

Puedes especificar el canal alfa (transparencia) y los canales RGB:

```
Dim colorTransparente As Color = Color.FromArgb(128, 255, 0, 0) '
Semitransparente rojo
```

- El primer valor (128) es el alfa: 0 (totalmente transparente) a 255 (opaco)
- Los otros valores corresponden a Rojo, Verde y Azul (0 a 255 cada uno)

---

5. Paleta de colores predefinidos

Visual Basic .NET incluye más de 100 colores predefinidos en la clase `Color`:

```
Dim color1 As Color = Color.Coral
Dim color2 As Color = Color.MediumPurple
```

Puedes explorar todos los colores con IntelliSense o usar una herramienta como el "Color Picker".

---

6. Uso en gráficos (GDI+)

Si estás trabajando con gráficos, puedes usar colores para pinceles y lápices:

```
Dim g As Graphics = Me.CreateGraphics()
Dim brush As New SolidBrush(Color.DarkCyan)
g.FillRectangle(brush, 10, 10, 100, 50)
```

---

7. Conversión entre colores y cadenas

Convertir un color a cadena:

```
Dim colorNombre As String = Color.Red.Name  ' "Red"
```

Obtener un color desde una cadena:

```
Dim colorDesdeNombre As Color = Color.FromName("Navy")
```

---

8. Usar un `ColorDialog` para elegir colores

```
Dim cd As New ColorDialog()
If cd.ShowDialog() = DialogResult.OK Then
    Me.BackColor = cd.Color
End If
```

---

9. Ejemplo completo

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        Dim cd As New ColorDialog()
        If cd.ShowDialog() = DialogResult.OK Then
            Me.BackColor = cd.Color
            Label1.ForeColor = cd.Color
        End If
    End Sub
End Class
```

---

10. Conclusión

Manejar colores en VB.NET es sencillo y permite una personalización extensa de las interfaces de usuario. Ya sea con colores predefinidos, personalizados con `ARGB`, o seleccionados con un `ColorDialog`, puedes enriquecer la experiencia visual de tus aplicaciones.

---

21. Abrir y Cerrar Formularios en Visual Basic .NET

Introducción

En Visual Basic .NET, trabajar con múltiples formularios es una tarea común al crear aplicaciones de escritorio con Windows Forms. Abrir y cerrar formularios correctamente permite una mejor navegación y control de la aplicación.

---

1. Crear y Abrir un Formulario

Para abrir un nuevo formulario desde otro formulario, primero debes asegurarte de que el nuevo formulario exista como una clase. Por ejemplo, supongamos que tienes un formulario llamado `Form2`.

Ejemplo de código para abrir `Form2` desde `Form1`:

```
' Dentro de Form1
Dim nuevoFormulario As New Form2()
nuevoFormulario.Show()
```

Explicación:

- `New Form2()` crea una nueva instancia del formulario.
- `.Show()` abre el formulario de manera **no modal**, es decir, puedes seguir interactuando con el formulario actual.

---

2. Abrir un Formulario de Modo Modal

Si deseas que el usuario no pueda interactuar con el formulario anterior hasta cerrar el nuevo, usa `.ShowDialog()`.

```
' Dentro de Form1
Dim ventanaModal As New Form2()
ventanaModal.ShowDialog()
```

Explicación:

- `.ShowDialog()` abre el formulario como **ventana modal**, bloqueando el acceso a otros formularios hasta que se cierre.

---

3. Cerrar un Formulario

Para cerrar un formulario desde su propio código:

```
Me.Close()
```

Explicación:

- `Me` se refiere al formulario actual.
- `.Close()` cierra el formulario.

---

4. Cerrar un Formulario desde Otro Formulario

Para cerrar otro formulario que has abierto:

```
Dim f As New Form2()
f.Show()

' Más adelante en el código
f.Close()
```

Debes tener una **referencia** al formulario que deseas cerrar.

---

5. Ocultar un Formulario en Lugar de Cerrar

Si no deseas cerrar un formulario sino solo ocultarlo temporalmente:

```
Me.Hide()
```

Esto es útil si planeas mostrarlo de nuevo más adelante sin volver a crear la instancia.

---

6. Ejemplo Completo

Supongamos que quieres que `Form1` abra `Form2`, y que al cerrar `Form2`, `Form1` vuelva a mostrarse:

Form1:

```
Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click
    Me.Hide()
    Dim f2 As New Form2()
    f2.ShowDialog()
    Me.Show()
End Sub
```

Form2:

```
Private Sub btnCerrar_Click(sender As Object, e As EventArgs) Handles
btnCerrar.Click
    Me.Close()
End Sub
```

---

7. Consejos Prácticos

- Evita usar `Application.Exit()` si solo quieres cerrar un formulario, ya que termina toda la aplicación.
- Usa `Using` si necesitas asegurar el cierre de formularios modales que consumen recursos:

```
Using f As New Form2()
    f.ShowDialog()
End Using
```

---

22. Uso de `Format` en Visual Basic .NET

🔹 ¿Qué es `Format`?

La función `Format` en Visual Basic .NET se utiliza para **dar formato a cadenas, números, fechas y horas** según un patrón específico, lo que facilita su presentación visual al usuario.

🔹 Sintaxis básica

```
Format(expression, [format], [firstDayOfWeek], [firstWeekOfYear])
```

- `expression`: El valor que se desea formatear.
- `format` (opcional): Una cadena que especifica el formato deseado.
- `firstDayOfWeek` y `firstWeekOfYear` (opcionales): Se usan con fechas, permiten ajustar el inicio de la semana o del año.

🔹 Ejemplos comunes

1. Formatear números

```
Dim valor As Double = 1234.5678
Console.WriteLine(Format(valor, "Standard"))      ' 1,234.57
Console.WriteLine(Format(valor, "Currency"))      ' $1,234.57
Console.WriteLine(Format(valor, "0.00"))          ' 1234.57
Console.WriteLine(Format(valor, "#,##0.00"))       ' 1,234.57
```

2. Formatear fechas

```
Dim fecha As Date = #6/26/2025 14:35:00#
Console.WriteLine(Format(fecha, "Short Date"))     ' 26/06/2025
Console.WriteLine(Format(fecha, "Long Date"))      ' jueves, 26 de junio de 2025
Console.WriteLine(Format(fecha, "dd-MM-yyyy"))     ' 26-06-2025
Console.WriteLine(Format(fecha, "hh:mm tt"))       ' 02:35 PM
```

3. Formatear como porcentaje

```
Dim porcentaje As Double = 0.8523
Console.WriteLine(Format(porcentaje, "Percent"))   ' 85.23%
```

🔹 Formatos predefinidos

Formato	Descripción	Ejemplo salida
`"General"`	Formato general para números	1234.57
`"Currency"`	Formato moneda local	$1,234.57 (depende cultura)
`"Fixed"`	Decimales fijos	1234.57
`"Standard"`	Con separador de miles	1,234.57
`"Percent"`	Convierte a porcentaje	85.23%
`"Short Date"`	Fecha corta	26/06/2025
`"Long Date"`	Fecha larga	jueves, 26 de junio de 2025
`"Time"`	Hora completa	14:35:00
`"yyyy-MM-dd"`	Personalizado de fecha ISO	2025-06-26

🔹 Consideraciones

- Si usas configuraciones regionales distintas, los formatos pueden variar (ej. símbolo de moneda, orden de fecha).
- Para formatos más complejos o globales, se recomienda usar `String.Format` o interpolación con `.ToString("formato")`.

---

23. Manejo de Archivos y Directorios en Visual Basic .NET

🔹 Introducción

El manejo de archivos y directorios en Visual Basic .NET se realiza principalmente mediante las clases del espacio de nombres `System.IO`. Estas clases permiten leer, escribir, copiar, mover y eliminar archivos y carpetas en el sistema.

🔹 Clases principales en `System.IO`

Clase	Función principal
`File`	Operaciones estáticas con archivos (crear, copiar, etc.)
`FileInfo`	Proporciona métodos para instancias de archivos
`Directory`	Operaciones estáticas con carpetas
`DirectoryInfo`	Métodos para carpetas como objetos
`StreamReader`	Lectura secuencial de archivos de texto
`StreamWriter`	Escritura secuencial en archivos de texto

🔹 Operaciones básicas con archivos

1. Crear un archivo de texto

```
Imports System.IO

File.WriteAllText("ejemplo.txt", "Hola mundo desde VB.NET")
```

2. Leer el contenido de un archivo

```
Dim contenido As String = File.ReadAllText("ejemplo.txt")
Console.WriteLine(contenido)
```

3. Agregar texto a un archivo existente

```
File.AppendAllText("ejemplo.txt", vbCrLf & "Otra línea de texto")
```

4. Copiar, mover y eliminar archivos

```
File.Copy("ejemplo.txt", "copia.txt", True)
File.Move("copia.txt", "nueva_ruta.txt")
File.Delete("nueva_ruta.txt")
```

🔹 Operaciones con directorios

1. Crear un directorio

```
Directory.CreateDirectory("CarpetaNueva")
```

2. Obtener archivos y subdirectorios

```
Dim archivos As String() = Directory.GetFiles("CarpetaNueva")
Dim carpetas As String() = Directory.GetDirectories("CarpetaNueva")
```

3. Mover y eliminar directorios

```
Directory.Move("CarpetaNueva", "OtraCarpeta")
Directory.Delete("OtraCarpeta", True) ' True para eliminar contenido
```

🔹 Lectura y escritura con `StreamReader` y `StreamWriter`

Escribir líneas en archivo:

```
Using escritor As New StreamWriter("salida.txt")
    escritor.WriteLine("Primera línea")
    escritor.WriteLine("Segunda línea")
End Using
```

Leer líneas de un archivo:

```
Using lector As New StreamReader("salida.txt")
    While Not lector.EndOfStream
        Console.WriteLine(lector.ReadLine())
    End While
End Using
```

🔹 Buenas prácticas

- Usa `Using` para asegurar el cierre automático de archivos.
- Controla excepciones con `Try...Catch` para manejar errores como archivos inexistentes.
- Verifica la existencia de archivos o directorios con `File.Exists` o `Directory.Exists` antes de operar.
- Evita rutas codificadas; utiliza `Path.Combine` para crear rutas seguras.

```
Dim rutaCompleta As String = Path.Combine("MisDocumentos", "archivo.txt")
```

Ejemplo completo: Crear, escribir y leer archivo

```
Imports System.IO

Module Program
    Sub Main()
        Dim ruta As String = "demo.txt"

        ' Crear archivo
        File.WriteAllText(ruta, "Hola desde VB.NET")

        ' Leer y mostrar
        Dim texto As String = File.ReadAllText(ruta)
        Console.WriteLine("Contenido: " & texto)

        ' Borrar archivo
        File.Delete(ruta)
    End Sub
End Module
```

---

24. Controles Comunes de Windows Forms en Visual Basic

Control	Descripción
`Button`	Botón para ejecutar una acción cuando se hace clic.
`Label`	Etiqueta para mostrar texto no editable.
`TextBox`	Caja de texto para que el usuario escriba datos.
`RichTextBox`	Caja de texto enriquecido (con formato, colores, etc.).
`CheckBox`	Casilla de verificación (activo/inactivo).
`RadioButton`	Botón de opción, usado en grupo para seleccionar una sola opción.
`ComboBox`	Lista desplegable de opciones con posibilidad de escritura.
`ListBox`	Lista de elementos visibles para seleccionar uno o varios.
`PictureBox`	Muestra imágenes.
`ProgressBar`	Barra de progreso visual.
`TrackBar`	Control deslizante (slider) para seleccionar un valor numérico.
`DateTimePicker`	Selector de fecha y hora.
`Timer`	Ejecuta acciones en intervalos de tiempo definidos.
`Panel`	Contenedor para agrupar controles.
`GroupBox`	Caja de grupo con bordes y título para agrupar controles.
`DataGridView`	Muestra datos en una tabla.
`TabControl`	Permite dividir el formulario en pestañas.
`TreeView`	Muestra elementos jerárquicos (como un árbol de carpetas).
`MenuStrip`	Crea menús (Archivo, Edición, etc.).
`ToolStrip`	Barra de herramientas con botones, listas, etc.
`OpenFileDialog`	Cuadro de diálogo para abrir archivos.
`SaveFileDialog`	Cuadro de diálogo para guardar archivos.
`ColorDialog`	Cuadro de diálogo para seleccionar un color.
`FontDialog`	Cuadro de diálogo para seleccionar fuente tipográfica.

Ejemplo Básico de Uso (Formulario con Botón y Etiqueta)

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        Label1.Text = "¡Hola, mundo!"
    End Sub
End Class
```

En este ejemplo, al hacer clic en `Button1`, el texto de `Label1` cambia.

Cómo Agregar Controles

En Visual Studio:

1. Abre el diseñador del formulario (doble clic en `Form1.vb`).
2. Usa el panel "Cuadro de herramientas" (Toolbox).
3. Arrastra los controles al formulario.
4. Cambia sus propiedades en la ventana "Propiedades".

---

25. ¿Qué es un evento?

Un evento es una **notificación** enviada por un objeto (el **emisor del evento**) para indicar que ha ocurrido una acción (como un clic de botón, una pulsación de tecla, etc.). El objeto que responde a este evento (el **manejador del evento**) contiene el código que se ejecutará.

🔹 Ejemplo básico de uso de eventos en VB.NET

Botón con evento `Click`:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        MessageBox.Show("¡Has hecho clic en el botón!")
    End Sub
End Class
```

- **Explicación**:
  - `Button1_Click` es un **manejador de eventos**.
  - `Handles Button1.Click` indica que este método se ejecutará cuando el botón `Button1` reciba el evento `Click`.

🔹 Declarar y usar tus propios eventos

Puedes declarar eventos personalizados en tus clases:

```
Public Class MiClase
    ' Declarar el evento
    Public Event AlCambiarNombre(ByVal nuevoNombre As String)

    Private _nombre As String

    Public Property Nombre As String
        Get
            Return _nombre
        End Get
        Set(value As String)
            _nombre = value
            ' Lanzar el evento
            RaiseEvent AlCambiarNombre(value)
        End Set
    End Property
End Class
```

Usar el evento en otra clase (por ejemplo, en un formulario):

```
Public Class Form1
    Private WithEvents obj As New MiClase

    Private Sub obj_AlCambiarNombre(nuevoNombre As String) Handles
obj.AlCambiarNombre
        MessageBox.Show("El nombre ha cambiado a: " & nuevoNombre)
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        obj.Nombre = "NuevoNombre"
    End Sub
End Class
```

🔹 Palabras clave importantes

Palabra clave	Significado
`Event`	Declara un evento.
`RaiseEvent`	Lanza (dispara) un evento.
`Handles`	Indica que un método maneja un evento.
`WithEvents`	Permite que una variable de objeto pueda manejar eventos.

🔹 Eventos comunes en controles de formularios

Control	Evento común	Descripción
Button	Click	Cuando se hace clic.
TextBox	TextChanged	Cuando cambia el texto.
Form	Load	Cuando se carga el formulario.
Timer	Tick	En cada intervalo del temporizador.

---

25. TIPOS DE MÉTODOS EN VB.NET

1. Sub (Subrutina)

No devuelve ningún valor.

```
Public Sub Saludar()
    Console.WriteLine("Hola, ¿cómo estás?")
End Sub
```

2. Function

Devuelve un valor.

```
Public Function Sumar(a As Integer, b As Integer) As Integer
    Return a + b
End Function
```

◆ LLAMAR A UN MÉTODO

```
Dim resultado As Integer = Sumar(3, 5)  ' Llama a una función
Saludar()                               ' Llama a una subrutina
```

◆ MÉTODOS CON PARÁMETROS OPCIONALES

```
Public Sub MostrarMensaje(Optional mensaje As String = "Hola por defecto")
    Console.WriteLine(mensaje)
End Sub
```

◆ MÉTODOS CON PARÁMETROS ByVal y ByRef

- `ByVal`: Pasa una copia del valor.
- `ByRef`: Pasa una referencia (modifica la variable original).

```
Public Sub CambiarValor(ByRef numero As Integer)
    numero = 100
End Sub
```

◆ MÉTODOS EN UNA CLASE

```
Public Class Calculadora
    Public Function Multiplicar(a As Integer, b As Integer) As Integer
        Return a * b
    End Function
End Class

' Uso
Dim calc As New Calculadora()
Dim resultado As Integer = calc.Multiplicar(4, 5)
```

MÉTODOS ESTÁTICOS (Shared)

```
Public Class Utilidades
    Public Shared Sub MostrarFechaActual()
        Console.WriteLine(DateTime.Now)
    End Sub
End Class

' Uso sin crear instancia
Utilidades.MostrarFechaActual()
```

---

27. 20 ejercicios con métodos en VB.NET

Ejercicio 1: Método Sub simple

Enunciado: Crea un método Sub llamado `MostrarSaludo` que imprima "¡Hola Mundo!" en consola.

```
Public Sub MostrarSaludo()
    Console.WriteLine("¡Hola Mundo!")
End Sub
```

Explicación: Método sin parámetros que solo muestra un texto.

Ejercicio 2: Método Function con parámetros

Enunciado: Crea una función `Sumar` que reciba dos números enteros y devuelva su suma.

```
Public Function Sumar(a As Integer, b As Integer) As Integer
    Return a + b
End Function
```

Ejercicio 3: Método con parámetro opcional

Enunciado: Crea un método `ImprimirMensaje` con un parámetro opcional `mensaje` que por defecto imprima "Mensaje predeterminado".

```
Public Sub ImprimirMensaje(Optional mensaje As String = "Mensaje
predeterminado")
    Console.WriteLine(mensaje)
End Sub
```

Ejercicio 4: Método que modifica un parámetro ByRef

Enunciado: Crea un método `Duplicar` que tome un parámetro entero ByRef y lo multiplique por 2.

```
Public Sub Duplicar(ByRef numero As Integer)
    numero *= 2
End Sub
```

Ejercicio 5: Método que devuelve un valor booleano

Enunciado: Crea una función `EsPar` que devuelva True si un número es par y False si es impar.

```
Public Function EsPar(numero As Integer) As Boolean
    Return (numero Mod 2 = 0)
End Function
```

Ejercicio 6: Método sin parámetros que devuelve la fecha actual

Enunciado: Crea una función `FechaActual` que devuelva la fecha y hora actual.

```
Public Function FechaActual() As DateTime
    Return DateTime.Now
End Function
```

Ejercicio 7: Método que imprime los números del 1 al 10

Enunciado: Crea un método `ImprimirNumeros` que imprima los números del 1 al 10 en consola.

```
Public Sub ImprimirNumeros()
    For i As Integer = 1 To 10
        Console.WriteLine(i)
    Next
End Sub
```

Ejercicio 8: Método con varios parámetros

Enunciado: Crea un método `CalcularAreaRectangulo` que reciba base y altura y devuelva el área.

```
Public Function CalcularAreaRectangulo(base As Double, altura As Double) As
Double
    Return base * altura
End Function
```

Ejercicio 9: Método Shared

Enunciado: Crea una clase `Matematicas` con un método Shared `Cuadrado` que devuelva el cuadrado de un número.

```
Public Class Matematicas
    Public Shared Function Cuadrado(num As Integer) As Integer
        Return num * num
    End Function
End Class
```

Ejercicio 10: Método que concatena cadenas

Enunciado: Crea un método `Concatenar` que reciba dos strings y devuelva su concatenación separada por un espacio.

```
Public Function Concatenar(str1 As String, str2 As String) As String
    Return str1 & " " & str2
End Function
```

Ejercicio 11: Método para verificar si un string está vacío

Enunciado: Crea una función `EsVacio` que devuelva True si el string recibido está vacío o solo tiene espacios.

```
Public Function EsVacio(texto As String) As Boolean
    Return String.IsNullOrWhiteSpace(texto)
End Function
```

Ejercicio 12: Método que recibe un array y devuelve el máximo

Enunciado: Crea una función `Maximo` que reciba un array de enteros y devuelva el mayor número.

```
Public Function Maximo(numeros() As Integer) As Integer
    Dim max As Integer = numeros(0)
    For Each num In numeros
        If num > max Then
            max = num
        End If
    Next
    Return max
End Function
```

Ejercicio 13: Método que devuelve la longitud de un string

Enunciado: Crea una función `LongitudCadena` que devuelva la longitud de un string.

```
Public Function LongitudCadena(texto As String) As Integer
    Return texto.Length
End Function
```

Ejercicio 14: Método que imprime un mensaje N veces

Enunciado: Crea un método `RepetirMensaje` que imprima un mensaje un número determinado de veces.

```
Public Sub RepetirMensaje(mensaje As String, veces As Integer)
    For i As Integer = 1 To veces
        Console.WriteLine(mensaje)
    Next
End Sub
```

Ejercicio 15: Método que calcula el factorial de un número

Enunciado: Crea una función `Factorial` que calcule el factorial de un número entero.

```
Public Function Factorial(n As Integer) As Integer
    Dim resultado As Integer = 1
    For i As Integer = 2 To n
        resultado *= i
    Next
    Return resultado
```

```
End Function
```

Ejercicio 16: Método que verifica si un año es bisiesto

Enunciado: Crea una función `EsBisiesto` que determine si un año es bisiesto.

```
Public Function EsBisiesto(anio As Integer) As Boolean
    Return (anio Mod 400 = 0) Or ((anio Mod 4 = 0) And (anio Mod 100 <> 0))
End Function
```

Ejercicio 17: Método que devuelve la suma de elementos de un array

Enunciado: Crea una función `SumarArray` que devuelva la suma de todos los números de un array.

```
Public Function SumarArray(numeros() As Integer) As Integer
    Dim suma As Integer = 0
    For Each num In numeros
        suma += num
    Next
    Return suma
End Function
```

Ejercicio 18: Método que convierte grados Celsius a Fahrenheit

Enunciado: Crea una función `CelsiusAFahrenheit` que reciba grados Celsius y devuelva Fahrenheit.

```
Public Function CelsiusAFahrenheit(celsius As Double) As Double
    Return (celsius * 9 / 5) + 32
End Function
```

Ejercicio 19: Método con múltiples parámetros opcionales

Enunciado: Crea un método `PresentarPersona` con parámetros opcionales para nombre, edad y ciudad.

```
Public Sub PresentarPersona(Optional nombre As String = "Desconocido", Optional
edad As Integer = 0, Optional ciudad As String = "Sin ciudad")
    Console.WriteLine($"Nombre: {nombre}, Edad: {edad}, Ciudad: {ciudad}")
End Sub
```

Ejercicio 20: Método que intercambia dos valores usando ByRef

Enunciado: Crea un método `Intercambiar` que intercambie dos números enteros.

```
Public Sub Intercambiar(ByRef a As Integer, ByRef b As Integer)
    Dim temp As Integer = a
    a = b
    b = temp
End Sub
```

¿Quieres que te prepare el código completo con un proyecto VB.NET o ejemplos para ejecutarlo en consola? ¿O prefieres ejercicios con soluciones explicadas?

28. Archivos

1. Archivo secuencial con registros (fichas) simples en VB.NET

Supongamos que tienes una ficha con estos campos:

- Id (Integer)
- Nombre (String)
- Edad (Integer)

Y quieres guardar varias fichas en un archivo de texto plano, cada línea un registro separado por comas (CSV simple).

Guardar registros (fichas) en archivo

```
Imports System.IO

Module Module1

    Sub Main()
        Dim rutaArchivo As String = "fichas.txt"

        ' Crear algunas fichas
        Dim fichas As New List(Of String)
        fichas.Add("1,Juan,25")
        fichas.Add("2,Ana,30")
        fichas.Add("3,Luis,22")

        ' Guardar fichas en archivo
        File.WriteAllLines(rutaArchivo, fichas)

        Console.WriteLine("Fichas guardadas en archivo.")
    End Sub

End Module
```

Leer fichas desde archivo

```
Imports System.IO

Module Module1

    Sub Main()
        Dim rutaArchivo As String = "fichas.txt"

        If File.Exists(rutaArchivo) Then
            Dim lineas() As String = File.ReadAllLines(rutaArchivo)

            For Each linea As String In lineas
                Dim campos() As String = linea.Split(","c)
                Dim id As Integer = Convert.ToInt32(campos(0))
                Dim nombre As String = campos(1)
                Dim edad As Integer = Convert.ToInt32(campos(2))

                Console.WriteLine($"Id: {id}, Nombre: {nombre}, Edad: {edad}")
            Next
        Else
            Console.WriteLine("El archivo no existe.")
        End If

    End Sub

End Module
```

---

2. Usar estructura o clase para la ficha

Puedes definir una clase o estructura para manejar mejor las fichas:

```
Public Class Ficha
    Public Property Id As Integer
    Public Property Nombre As String
    Public Property Edad As Integer

    Public Overrides Function ToString() As String
        Return $"{Id},{Nombre},{Edad}"
    End Function
End Class
```

Y luego usar esta clase para leer y escribir:

```
Imports System.IO

Module Module1

    Sub Main()
        Dim rutaArchivo As String = "fichas.txt"

        ' Crear lista de fichas
        Dim listaFichas As New List(Of Ficha) From {
            New Ficha With {.Id = 1, .Nombre = "Juan", .Edad = 25},
            New Ficha With {.Id = 2, .Nombre = "Ana", .Edad = 30},
            New Ficha With {.Id = 3, .Nombre = "Luis", .Edad = 22}
        }

        ' Guardar fichas en archivo
        Dim lineas As New List(Of String)
        For Each f As Ficha In listaFichas
            lineas.Add(f.ToString())
        Next
        File.WriteAllLines(rutaArchivo, lineas)

        ' Leer fichas desde archivo
        If File.Exists(rutaArchivo) Then
            Dim fichasLeidas As New List(Of Ficha)
            Dim lineasLeidas() As String = File.ReadAllLines(rutaArchivo)

            For Each linea As String In lineasLeidas
                Dim campos() As String = linea.Split(","c)
                Dim ficha As New Ficha With {
                    .Id = Convert.ToInt32(campos(0)),
                    .Nombre = campos(1),
                    .Edad = Convert.ToInt32(campos(2))
                }
                fichasLeidas.Add(ficha)
            Next

            ' Mostrar fichas
            For Each f As Ficha In fichasLeidas
                Console.WriteLine($"Id: {f.Id}, Nombre: {f.Nombre}, Edad:
{f.Edad}")
            Next
        End If

    End Sub

End Module
```

3. Otras opciones para archivos ficha

- Puedes usar archivos binarios para guardar registros estructurados con más eficiencia (requiere manejo con `BinaryWriter` y `BinaryReader`).
- Para proyectos más grandes, puedes usar bases de datos SQLite o XML para guardar fichas.
- También existe el archivo Random Access, pero es menos usado en VB.NET moderno.

Ejemplo: Guardar y cargar una lista de strings en un archivo de texto

1. Guardar lista en archivo

```
Imports System.IO

Module Module1
    Sub Main()
        ' Crear una lista de ejemplo
        Dim lista As New List(Of String) From {
            "Elemento 1",
            "Elemento 2",
            "Elemento 3"
        }

        ' Ruta del archivo
        Dim rutaArchivo As String = "lista.txt"

        ' Guardar lista en archivo
        File.WriteAllLines(rutaArchivo, lista)

        Console.WriteLine("Lista guardada en archivo.")
    End Sub
End Module
```

- `File.WriteAllLines` guarda cada elemento de la lista como una línea en el archivo.

2. Leer lista desde archivo

```
Imports System.IO

Module Module1
    Sub Main()
        Dim rutaArchivo As String = "lista.txt"

        ' Leer todas las líneas del archivo en una lista
        Dim listaDesdeArchivo As List(Of String) =
File.ReadAllLines(rutaArchivo).ToList()

        ' Mostrar elementos leídos
        For Each elemento As String In listaDesdeArchivo
            Console.WriteLine(elemento)
        Next
    End Sub
End Module
```

- `File.ReadAllLines` lee todas las líneas y devuelve un arreglo, que convertimos a lista con `.ToList()`.

Manejo básico de archivos con listas en VB.NET

- Para **guardar** una lista simple en un archivo de texto, usa `File.WriteAllLines(ruta, lista)`.
- Para **leer** una lista desde un archivo, usa `File.ReadAllLines(ruta).ToList()`.

---

28. 20 ejercicios de archivos en Visual Basic .NET

Ejercicio 1: Crear y escribir texto en un archivo

Enunciado:
Crear un programa que cree un archivo de texto llamado `archivo1.txt` y escriba en él la frase: "Hola, Visual Basic .NET".

Explicación:
Usamos `StreamWriter` para crear el archivo y escribir texto en él.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using sw As New StreamWriter("archivo1.txt")
            sw.WriteLine("Hola, Visual Basic .NET")
        End Using
        Console.WriteLine("Archivo creado y texto escrito.")
    End Sub
End Module
```

Ejercicio 2: Leer y mostrar contenido de un archivo

Enunciado:
Leer el contenido del archivo `archivo1.txt` y mostrarlo por consola.

Explicación:
Usamos `StreamReader` para leer línea por línea el contenido del archivo.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using sr As New StreamReader("archivo1.txt")
            Dim line As String
            line = sr.ReadLine()
            While line IsNot Nothing
                Console.WriteLine(line)
                line = sr.ReadLine()
            End While
        End Using
    End Sub
End Module
```

Ejercicio 3: Añadir texto a un archivo existente

Enunciado:
Agregar al final de `archivo1.txt` la línea "Esta es una línea adicional".

Explicación:
Abrimos el archivo en modo append para añadir contenido sin borrar lo existente.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using sw As New StreamWriter("archivo1.txt", True)
            sw.WriteLine("Esta es una línea adicional")
        End Using
        Console.WriteLine("Texto añadido al archivo.")
    End Sub
End Module
```

Ejercicio 4: Contar líneas de un archivo

Enunciado:
Contar y mostrar el número total de líneas que tiene `archivo1.txt`.

Explicación:
Leemos línea a línea y contamos cuántas hay.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim count As Integer = 0
        Using sr As New StreamReader("archivo1.txt")
            While sr.ReadLine() IsNot Nothing
                count += 1
            End While
        End Using
        Console.WriteLine("El archivo tiene " & count & " líneas.")
    End Sub
End Module
```

Ejercicio 5: Buscar una palabra en un archivo

Enunciado:
Buscar si la palabra "Visual" está en `archivo1.txt` y mostrar un mensaje según el resultado.

Explicación:
Leemos cada línea y usamos `Contains` para buscar la palabra.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim found As Boolean = False
        Using sr As New StreamReader("archivo1.txt")
            Dim line As String = sr.ReadLine()
            While line IsNot Nothing And Not found
                If line.Contains("Visual") Then
                    found = True
                End If
                line = sr.ReadLine()
            End While
        End Using

        If found Then
            Console.WriteLine("La palabra 'Visual' fue encontrada.")
        Else
            Console.WriteLine("La palabra 'Visual' no está en el archivo.")
        End If
    End Sub
End Module
```

Ejercicio 6: Copiar archivo

Enunciado:
Copiar el archivo `archivo1.txt` a `archivo_copia.txt`.

Explicación:
Usamos el método `File.Copy` para copiar el archivo.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        File.Copy("archivo1.txt", "archivo_copia.txt", True)
        Console.WriteLine("Archivo copiado correctamente.")
    End Sub
End Module
```

Ejercicio 7: Eliminar un archivo

Enunciado:
Eliminar el archivo `archivo_copia.txt` si existe.

Explicación:
Verificamos si existe el archivo y luego lo eliminamos con `File.Delete`.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        If File.Exists("archivo_copia.txt") Then
            File.Delete("archivo_copia.txt")
            Console.WriteLine("Archivo eliminado.")
        Else
            Console.WriteLine("El archivo no existe.")
        End If
    End Sub
End Module
```

Ejercicio 8: Leer archivo completo en una sola cadena

Enunciado:
Leer todo el contenido de `archivo1.txt` en una sola variable y mostrarlo.

Explicación:
Usamos `File.ReadAllText` para leer todo de una vez.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim contenido As String = File.ReadAllText("archivo1.txt")
        Console.WriteLine(contenido)
    End Sub
End Module
```

Ejercicio 9: Guardar un array de strings en un archivo

Enunciado:
Guardar en `archivo_array.txt` las líneas: "Línea 1", "Línea 2" y "Línea 3".

Explicación:
Usamos `File.WriteAllLines` que escribe todas las líneas del array en el archivo.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim lineas() As String = {"Línea 1", "Línea 2", "Línea 3"}
        File.WriteAllLines("archivo_array.txt", lineas)
        Console.WriteLine("Archivo creado con varias líneas.")
    End Sub
End Module
```

Ejercicio 10: Leer todas las líneas en un array y mostrarlas

Enunciado:
Leer todas las líneas de `archivo_array.txt` en un array y mostrarlas en consola.

Explicación:
Usamos `File.ReadAllLines` para obtener un array de líneas.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim lineas() As String = File.ReadAllLines("archivo_array.txt")
        For Each linea As String In lineas
            Console.WriteLine(linea)
        Next
    End Sub
End Module
```

Ejercicio 11: Crear un archivo binario con números enteros

Enunciado:
Crear un archivo binario llamado `datos.bin` que contenga los números 10, 20 y 30.

Explicación:
Usamos `BinaryWriter` para escribir datos binarios.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using bw As New BinaryWriter(File.Open("datos.bin", FileMode.Create))
            bw.Write(10)
            bw.Write(20)
            bw.Write(30)
        End Using
        Console.WriteLine("Archivo binario creado con números.")
    End Sub
End Module
```

Ejercicio 12: Leer números enteros de un archivo binario

Enunciado:
Leer los números enteros almacenados en `datos.bin` y mostrarlos.

Explicación:
Usamos `BinaryReader` para leer los enteros; leemos hasta el final del archivo.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using br As New BinaryReader(File.Open("datos.bin", FileMode.Open))
            While br.BaseStream.Position < br.BaseStream.Length
                Dim numero As Integer = br.ReadInt32()
                Console.WriteLine(numero)
            End While
        End Using
    End Sub
End Module
```

Ejercicio 13: Guardar una lista de objetos en archivo texto

Enunciado:
Guardar una lista de personas (nombre y edad) en un archivo `personas.txt`, cada persona en una línea con formato "nombre,edad".

Explicación:
Convertimos la lista a cadenas con formato y las escribimos.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim personas As New List(Of Tuple(Of String, Integer)) From {
            Tuple.Create("Ana", 25),
            Tuple.Create("Luis", 30),
            Tuple.Create("Marta", 22)
        }

        Using sw As New StreamWriter("personas.txt")
            For Each p In personas
                sw.WriteLine(p.Item1 & "," & p.Item2)
            Next
        End Using

        Console.WriteLine("Personas guardadas en archivo.")
    End Sub
End Module
```

Ejercicio 14: Leer archivo de personas y mostrar la más joven

Enunciado:
Leer el archivo `personas.txt`, parsear nombre y edad, y mostrar el nombre de la persona más joven.

Explicación:
Leemos línea a línea, dividimos por coma, comparamos edades.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim nombreJoven As String = ""
        Dim edadMin As Integer = Integer.MaxValue

        Using sr As New StreamReader("personas.txt")
            Dim line As String = sr.ReadLine()
            While line IsNot Nothing
                Dim partes() As String = line.Split(","c)
                Dim nombre As String = partes(0)
                Dim edad As Integer = Integer.Parse(partes(1))
                If edad < edadMin Then
                    edadMin = edad
                    nombreJoven = nombre
                End If
                line = sr.ReadLine()
            End While
        End Using

        Console.WriteLine("La persona más joven es: " & nombreJoven & " con " &
edadMin & " años.")
    End Sub
End Module
```

Ejercicio 15: Guardar y leer datos usando `FileStream`

Enunciado:
Crear un archivo `datos.dat` y escribir un arreglo de bytes `[1,2,3,4,5]`. Luego leerlo y mostrar los bytes.

Explicación:
Usamos `FileStream` directamente para escribir y leer bytes.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim datos() As Byte = {1, 2, 3, 4, 5}

        ' Escritura
        Using fs As New FileStream("datos.dat", FileMode.Create)
            fs.Write(datos, 0, datos.Length)
        End Using

        ' Lectura
        Using fs As New FileStream("datos.dat", FileMode.Open)
            Dim buffer(fs.Length - 1) As Byte
            fs.Read(buffer, 0, buffer.Length)
            Console.WriteLine("Bytes leídos:")
            For Each b As Byte In buffer
                Console.Write(b & " ")
            Next
            Console.WriteLine()
        End Using
    End Sub
End Module
```

Ejercicio 16: Crear un archivo CSV con encabezados

Enunciado:
Crear un archivo `productos.csv` con columnas "Producto", "Precio" y "Cantidad". Agregar 3 productos.

Explicación:
Escribimos la primera línea con encabezados y luego líneas con datos separados por comas.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Using sw As New StreamWriter("productos.csv")
            sw.WriteLine("Producto,Precio,Cantidad")
            sw.WriteLine("Manzana,0.5,10")
            sw.WriteLine("Pan,1.2,5")
            sw.WriteLine("Leche,0.8,8")
        End Using
        Console.WriteLine("Archivo CSV creado.")
    End Sub
End Module
```

Ejercicio 17: Leer archivo CSV y calcular total

Enunciado:
Leer `productos.csv` y calcular el total (Precio * Cantidad) de todos los productos.

Explicación:
Leemos línea a línea ignorando la primera, parseamos precio y cantidad y calculamos suma.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim total As Double = 0
        Using sr As New StreamReader("productos.csv")
            Dim linea As String = sr.ReadLine() ' Salta encabezado
            linea = sr.ReadLine()
            While linea IsNot Nothing
                Dim partes() As String = linea.Split(","c)
                Dim precio As Double = Double.Parse(partes(1))
                Dim cantidad As Integer = Integer.Parse(partes(2))
                total += precio * cantidad
                linea = sr.ReadLine()
            End While
        End Using
        Console.WriteLine("El total de la compra es: " & total.ToString("F2"))
    End Sub
End Module
```

Ejercicio 18: Actualizar línea en un archivo de texto

Enunciado:
Actualizar la línea que contiene "Pan" en `productos.csv` para cambiar la cantidad a 7.

Explicación:
Leemos todo, modificamos la línea y sobreescribimos el archivo.

Código:

```
Imports System.IO

Module Module1
    Sub Main()
        Dim lineas As List(Of String) =
File.ReadAllLines("productos.csv").ToList()
        For i As Integer = 0 To lineas.Count - 1
            If lineas(i).StartsWith("Pan,") Then
                Dim partes() As String = lineas(i).Split(","c)
                partes(2) = "7" ' Cambiar cantidad
                lineas(i) = String.Join(",", partes)
            End If
        Next
        File.WriteAllLines("productos.csv", lineas)
        Console.WriteLine("Archivo actualizado.")
    End Sub
End Module
```

Ejercicio 19: Borrar línea que contiene un texto en archivo

Enunciado:
Eliminar la línea que contiene "Leche" de `productos.csv`.

Explicación:
Filtramos todas las líneas que no contienen "Leche" y reescribimos el archivo.

Código:

```
Imports System.IO
Imports System.Linq

Module Module1
    Sub Main()
        Dim lineas As List(Of String) =
File.ReadAllLines("productos.csv").ToList()
        Dim nuevasLineas = lineas.Where(Function(l) Not
l.Contains("Leche")).ToList()
        File.WriteAllLines("productos.csv", nuevasLineas)
        Console.WriteLine("Línea con 'Leche' eliminada.")
    End Sub
End Module
```

Ejercicio 20: Manejo de archivos con diálogo para seleccionar archivo

Enunciado:
Crear un programa con Windows Forms que permita seleccionar un archivo de texto y mostrar su contenido en un `TextBox`.

Explicación:
Usamos `OpenFileDialog` para seleccionar el archivo y `File.ReadAllText` para mostrarlo.

Código:

```
Imports System.IO
Imports System.Windows.Forms

Public Class Form1
    Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click
        Using ofd As New OpenFileDialog()
            ofd.Filter = "Archivos de texto (*.txt)|*.txt|Todos los archivos
(*.*)|*.*"
            If ofd.ShowDialog() = DialogResult.OK Then
                Dim contenido As String = File.ReadAllText(ofd.FileName)
                txtContenido.Text = contenido
            End If
        End Using
    End Sub
End Class
```

Nota: Este último ejemplo es para un proyecto Windows Forms con un botón `btnAbrir` y un `TextBox` llamado `txtContenido`.

Ejercicios de archivos de ficha

🟢 Nivel Básico (1–7)

1. Crear una ficha de agenda con nombre y teléfono

Enunciado: Crea un programa que permita ingresar el nombre y el teléfono de una persona y los muestre en un cuadro de mensaje.

Solución:

```
Module Agenda
    Sub Main()
        Console.Write("Nombre: ")
        Dim nombre As String = Console.ReadLine()
        Console.Write("Teléfono: ")
        Dim telefono As String = Console.ReadLine()
        Console.WriteLine("Ficha creada: " & nombre & " - " & telefono)
    End Sub
End Module
```

Explicación: Se captura entrada del usuario por consola y se imprime como ficha simple.

---

2. Validar que el teléfono tenga solo números

Enunciado: Modifica el programa anterior para que el teléfono solo acepte dígitos.

Solución:

```
Module Agenda
    Sub Main()
        Console.Write("Nombre: ")
        Dim nombre As String = Console.ReadLine()
        Dim telefono As String
        Do
            Console.Write("Teléfono (solo dígitos): ")
            telefono = Console.ReadLine()
        Loop While Not IsNumeric(telefono)
        Console.WriteLine("Ficha creada: " & nombre & " - " & telefono)
    End Sub
End Module
```

Explicación: Se usa `IsNumeric` en un bucle para validar la entrada.

---

3. Guardar múltiples fichas en un arreglo

Enunciado: Permite ingresar 3 fichas y mostrarlas todas.

Solución:

```
Module Agenda
    Sub Main()
        Dim nombres(2) As String
        Dim telefonos(2) As String
        For i As Integer = 0 To 2
            Console.Write("Nombre " & (i + 1) & ": ")
            nombres(i) = Console.ReadLine()
            Console.Write("Teléfono " & (i + 1) & ": ")
            telefonos(i) = Console.ReadLine()
        Next
        Console.WriteLine("Listado de fichas:")
        For i As Integer = 0 To 2
            Console.WriteLine(nombres(i) & " - " & telefonos(i))
        Next
    End Sub
End Module
```

Explicación: Usa dos arreglos para guardar y mostrar las fichas.

---

4. Mostrar ficha por índice

Enunciado: Agrega la opción de mostrar una ficha según su número.

Solución:

```
Module Agenda
    Sub Main()
        Dim nombres(2) As String
        Dim telefonos(2) As String
        For i As Integer = 0 To 2
            Console.Write("Nombre " & (i + 1) & ": ")
            nombres(i) = Console.ReadLine()
            Console.Write("Teléfono " & (i + 1) & ": ")
            telefonos(i) = Console.ReadLine()
        Next
        Console.Write("Mostrar ficha número (1-3): ")
        Dim indice As Integer = Integer.Parse(Console.ReadLine()) - 1
        Console.WriteLine("Ficha: " & nombres(indice) & " - " &
telefonos(indice))
    End Sub
End Module
```

Explicación: Se accede al arreglo mediante el índice proporcionado por el usuario.

---

5. Usar estructura para representar fichas

Enunciado: Define una estructura `Ficha` con nombre y teléfono.

Solución:

```
Module Agenda
    Structure Ficha
        Dim nombre As String
        Dim telefono As String
    End Structure

    Sub Main()
        Dim f As Ficha
        Console.Write("Nombre: ")
        f.nombre = Console.ReadLine()
        Console.Write("Teléfono: ")
        f.telefono = Console.ReadLine()
        Console.WriteLine("Ficha: " & f.nombre & " - " & f.telefono)
    End Sub
End Module
```

Explicación: Se introduce `Structure` para agrupar los datos.

---

32 Controles de Interfaz Gráfica en Windows Forms – Visual Basic .NET

🎯 Objetivos de la lección

- Conocer los controles básicos de Windows Forms.
- Comprender cómo agregar y configurar controles en el diseñador.
- Aprender a programar eventos asociados a los controles.
- Crear una interfaz funcional que interactúe con el usuario.

🧩 1. Introducción a Windows Forms

Windows Forms es una tecnología de Microsoft que permite crear aplicaciones de escritorio con interfaces gráficas. En Visual Basic .NET, puedes diseñar formularios visualmente y agregar controles como botones, cuadros de texto, etiquetas, listas, entre otros.

¿Dónde encontrar los controles?

En el entorno de desarrollo **Visual Studio**, al crear un proyecto de tipo *Windows Forms App (.NET Framework)*, se habilita una caja de herramientas (Toolbox) con todos los controles.

🧱 2. Controles Básicos y su Uso

2.1 Label (Etiqueta)

- **Propósito**: Mostrar texto estático.
- **Propiedades clave**:
    - `Text`: El texto mostrado.
    - `Font`: Fuente del texto.
    - `ForeColor`: Color del texto.

```
Label1.Text = "Bienvenido al programa"
```

2.2 TextBox (Cuadro de texto)

- **Propósito**: Permitir la entrada de texto.
- **Propiedades clave**:
    - `Text`: El texto ingresado.

- `Multiline`: Permite varias líneas.
- `PasswordChar`: Para contraseñas.

```
Dim nombre As String = TextBox1.Text
```

2.3 Button (Botón)

- **Propósito**: Ejecutar acciones al hacer clic.
- **Propiedades clave**:
  - `Text`: Texto en el botón.
- **Evento importante**:
  - `Click`

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    MessageBox.Show("Hola, " & TextBox1.Text)
End Sub
```

2.4 CheckBox (Casilla de verificación)

- **Propósito**: Selección múltiple de opciones.
- **Propiedades clave**:
  - `Checked`: True o False.

```
If CheckBox1.Checked Then
    MessageBox.Show("Aceptaste los términos.")
End If
```

2.5 RadioButton (Botón de opción)

- **Propósito**: Selección exclusiva entre opciones.
- **Uso común**: Agrupados en un **GroupBox**.

```
If RadioButton1.Checked Then
    MessageBox.Show("Opción A seleccionada")
End If
```

2.6 ComboBox (Lista desplegable)

- **Propósito**: Selección de una opción entre varias.
- **Métodos comunes**:
  - `.Items.Add("Opción")`

- `.SelectedItem`

```
ComboBox1.Items.Add("España")
Dim paisSeleccionado As String = ComboBox1.SelectedItem.ToString()
```

2.7 ListBox (Lista de elementos)

- **Propósito**: Lista de selección.
- **Diferencia con ComboBox**: muestra todas las opciones visibles.

```
ListBox1.Items.Add("Elemento 1")
Dim seleccionado As String = ListBox1.SelectedItem.ToString()
```

2.8 PictureBox (Imagen)

- **Propósito**: Mostrar imágenes.
- **Propiedades clave**:
  - `Image`: Imagen mostrada.
  - `SizeMode`: Ajuste de la imagen.

```
PictureBox1.Image = Image.FromFile("C:\ruta\imagen.jpg")
```

2.9 DateTimePicker (Selector de fecha)

- **Propósito**: Permite seleccionar una fecha.
- **Propiedad**:
  - `.Value`: Devuelve un objeto Date.

```
Dim fechaSeleccionada As Date = DateTimePicker1.Value
```

2.10 Timer (Temporizador)

- **Propósito**: Ejecutar código cada cierto tiempo.
- **Eventos**:
  - `.Tick`

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
    LabelHora.Text = Now.ToString("hh:mm:ss")
End Sub
```

👩‍💻 3. Ejemplo Práctico: Formulario de Registro

Diseño

Controles a usar:

- `Label`: “Nombre”, “Sexo”, “Fecha de nacimiento”
- `TextBox`: Para nombre
- `RadioButton`: “Masculino”, “Femenino”
- `DateTimePicker`: Para la fecha
- `Button`: “Registrar”

Código del botón

```
Private Sub btnRegistrar_Click(sender As Object, e As EventArgs) Handles
btnRegistrar.Click
    Dim nombre As String = txtNombre.Text
    Dim sexo As String = If(rbtnMasculino.Checked, "Masculino", "Femenino")
    Dim fecha As Date = dtpNacimiento.Value

    MessageBox.Show("Nombre: " & nombre & vbCrLf &
                    "Sexo: " & sexo & vbCrLf &
                    "Nacimiento: " & fecha.ToShortDateString())
End Sub
```

⚙ 4. Buenas Prácticas

- **Nombra bien los controles**: En lugar de `TextBox1`, usa `txtNombre`.
- **Agrupa controles lógicamente**: Usa `GroupBox` para organizar opciones.
- **Valida entradas**: Usa `If` para comprobar que el usuario llenó todo.
- **Evita repetir código**: Usa funciones para reutilizar lógica.

🧠 5. Actividades Propuestas

1. Crea un formulario con los datos personales: nombre, edad, género, país.
2. Implementa un sistema de encuestas donde se seleccionan varias opciones.
3. Haz un programa que muestre la hora actual usando `Timer`.

📝 6. Resumen

- Los **controles de Windows Forms** permiten crear interfaces amigables.

- Cada control tiene propiedades, eventos y métodos específicos.
- El manejo de eventos como `Click`, `CheckedChanged`, `SelectedIndexChanged` permite crear aplicaciones interactivas.

---

33. Ejercicios Form

Programas de dificultad facil (1–20)

1. Mostrar mensaje al hacer clic en un botón

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    MessageBox.Show("Hola mundo")

End Sub
```

Explicación:
Muestra un mensaje emergente con el texto "Hola mundo" cuando se hace clic en el botón.

---

2. Copiar texto de TextBox a Label

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Text = TextBox1.Text

End Sub
```

Explicación:
Copia el texto ingresado en el TextBox y lo muestra en el Label.

---

3. Suma de dos números

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim suma As Double = Val(TextBox1.Text) + Val(TextBox2.Text)

    Label1.Text = "Suma: " & suma

End Sub
```

Explicación:
Convierte los valores a numéricos y muestra su suma en el Label.

---

4. Cambiar color de fondo

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Me.BackColor = Color.LightGreen

End Sub
```

Explicación:
Cambia el color del fondo del formulario a verde claro.

---

5. Mostrar opción de CheckBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    If CheckBox1.Checked Then
        Label1.Text = "Seleccionado"
    Else
        Label1.Text = "No seleccionado"
    End If
End Sub
```

Explicación:
Verifica si el CheckBox está marcado y muestra el estado.

---

6. Seleccionar género con RadioButton

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    If RadioButton1.Checked Then
        Label1.Text = "Masculino"
    ElseIf RadioButton2.Checked Then
        Label1.Text = "Femenino"
    End If
End Sub
```

Explicación:
Muestra el texto correspondiente al RadioButton seleccionado.

---

7. Mostrar fecha y hora actual

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Label1.Text = DateTime.Now.ToString()
End Sub
```

Explicación:
Muestra la fecha y hora actual en el Label.

---

8. Agregar ítem a ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    ListBox1.Items.Add(TextBox1.Text)

End Sub
```

Explicación:
Agrega el texto del TextBox como nuevo elemento del ListBox.

---

9. Mostrar elemento seleccionado en ComboBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Text = "Seleccionado: " & ComboBox1.Text

End Sub
```

Explicación:
Muestra cuál opción del ComboBox ha sido seleccionada.

---

10. Limpiar campos

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    TextBox1.Clear()

    TextBox2.Clear()

    Label1.Text = ""

End Sub
```

Explicación:
Borra el contenido de dos TextBox y un Label.

Perfecto, aquí tienes los **programas del 11 al 20**, todos de **nivel fácil**, con su código y explicación.

---

11. Contador de clics

```
Dim contador As Integer = 0


Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    contador += 1

    Label1.Text = "Clics: " & contador

End Sub
```

Explicación:
Cada vez que se pulsa el botón, se incrementa el contador y se muestra en un Label.

---

12. Cambiar texto de Label con ComboBox

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBox1.SelectedIndexChanged

    Label1.Text = "Has seleccionado: " & ComboBox1.SelectedItem.ToString()

End Sub
```

Explicación:
Cuando el usuario cambia la selección del ComboBox, se muestra en el Label.

---

13. Calcular área de un rectángulo

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim base As Double = Val(TextBox1.Text)

    Dim altura As Double = Val(TextBox2.Text)

    Dim area As Double = base * altura

    Label1.Text = "Área: " & area

End Sub
```

Explicación:
Toma base y altura desde los TextBox y muestra el área calculada.

---

14. Activar/Desactivar botón con CheckBox

```
Private Sub CheckBox1_CheckedChanged(sender As Object, e As EventArgs) Handles
CheckBox1.CheckedChanged

    Button1.Enabled = CheckBox1.Checked

End Sub
```

Explicación:
El botón se activa o desactiva dependiendo del estado del CheckBox.

---

15. Concatenar nombre y apellido

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim nombreCompleto As String = TextBox1.Text & " " & TextBox2.Text

    Label1.Text = "Nombre completo: " & nombreCompleto

End Sub
```

Explicación:
Concatena el contenido de dos TextBox y lo muestra como nombre completo.

---

16. Cambiar tamaño de fuente

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Font = New Font("Arial", 16, FontStyle.Bold)

End Sub
```

Explicación:
Modifica el tamaño y estilo de fuente del Label.

---

17. Mostrar número aleatorio

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim rnd As New Random()

    Dim numero As Integer = rnd.Next(1, 101)

    Label1.Text = "Número aleatorio: " & numero

End Sub
```

Explicación:
Genera un número aleatorio del 1 al 100 y lo muestra.

---

18. Calcular el cuadrado de un número

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim numero As Double = Val(TextBox1.Text)

    Label1.Text = "Cuadrado: " & (numero ^ 2)

End Sub
```

Explicación:
Eleva al cuadrado el número ingresado y muestra el resultado.

---

19. Convertir texto a mayúsculas

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Text = TextBox1.Text.ToUpper()

End Sub
```

Explicación:
Convierte el texto del TextBox a mayúsculas y lo muestra.

---

20. Mostrar cantidad de caracteres

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim longitud As Integer = TextBox1.Text.Length

    Label1.Text = "Caracteres: " & longitud

End Sub
```

Explicación:
Muestra cuántos caracteres tiene el texto introducido en el TextBox.

Programas de dificultad media (21–40)

21. Validar entrada numérica en TextBox

```
Private Sub TextBox1_KeyPress(sender As Object, e As KeyPressEventArgs) Handles
TextBox1.KeyPress

    If Not Char.IsDigit(e.KeyChar) AndAlso Not Char.IsControl(e.KeyChar) Then

        e.Handled = True

    End If

End Sub
```

Explicación:
Evita que el usuario escriba caracteres no numéricos en el TextBox.

---

22. Mostrar lista ordenada en ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim items As New List(Of String)

    For Each item As String In ListBox1.Items

        items.Add(item)

    Next

    items.Sort()

    ListBox1.Items.Clear()

    ListBox1.Items.AddRange(items.ToArray())

End Sub
```

Explicación:
Ordena alfabéticamente los elementos del ListBox al hacer clic.

---

23. Contar palabras en TextBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim palabras() As String = TextBox1.Text.Split(New Char() {" "c},
StringSplitOptions.RemoveEmptyEntries)

    Label1.Text = "Palabras: " & palabras.Length

End Sub
```

Explicación:
Cuenta y muestra el número de palabras escritas en el TextBox.

---

24. Mostrar imagen en PictureBox desde archivo

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Using ofd As New OpenFileDialog()
        ofd.Filter = "Archivos de imagen|*.jpg;*.png;*.bmp"
        If ofd.ShowDialog() = DialogResult.OK Then
            PictureBox1.Image = Image.FromFile(ofd.FileName)
        End If
    End Using
End Sub
```

Explicación:
Permite seleccionar una imagen desde disco y mostrarla en un PictureBox.

---

25. Habilitar/Deshabilitar controles con RadioButtons

```
Private Sub RadioButton1_CheckedChanged(sender As Object, e As EventArgs)
Handles RadioButton1.CheckedChanged, RadioButton2.CheckedChanged
    TextBox1.Enabled = RadioButton1.Checked
    TextBox2.Enabled = RadioButton2.Checked
End Sub
```

Explicación:
Activa un TextBox u otro según el RadioButton seleccionado.

---

26. Mostrar ítems seleccionados en ListBox múltiple

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim seleccionados As String = ""
    For Each item As String In ListBox1.SelectedItems
        seleccionados &= item & ", "
    Next
    If seleccionados.Length > 2 Then
        seleccionados = seleccionados.Substring(0, seleccionados.Length - 2)
    End If
    Label1.Text = "Seleccionados: " & seleccionados
End Sub
```

Explicación:
Concatena y muestra los elementos seleccionados en un ListBox con selección múltiple.

---

27. Calcular IMC (Índice de Masa Corporal)

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim peso As Double = Val(TextBox1.Text)

    Dim altura As Double = Val(TextBox2.Text)

    If altura > 0 Then

        Dim imc As Double = peso / (altura * altura)

        Label1.Text = "IMC: " & imc.ToString("F2")

    Else

        MessageBox.Show("Altura debe ser mayor que cero.")

    End If

End Sub
```

Explicación:
Calcula y muestra el IMC dado peso y altura ingresados.

---

28. Copiar texto con formato RichTextBox a TextBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    TextBox1.Text = RichTextBox1.Text

End Sub
```

Explicación:
Copia texto plano desde un RichTextBox a un TextBox.

---

29. Agregar fecha actual a ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    ListBox1.Items.Add(DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss"))

End Sub
```

Explicación:
Agrega la fecha y hora actual como nuevo elemento en un ListBox.

---

30. Mostrar índice del ítem seleccionado en ComboBox

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBox1.SelectedIndexChanged

    Label1.Text = "Índice seleccionado: " & ComboBox1.SelectedIndex

End Sub
```

Explicación:
Muestra el índice del elemento seleccionado en un ComboBox.

31. Calcular promedio de números ingresados en TextBoxes

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim n1 As Double = Val(TextBox1.Text)
    Dim n2 As Double = Val(TextBox2.Text)
    Dim n3 As Double = Val(TextBox3.Text)
    Dim promedio As Double = (n1 + n2 + n3) / 3
    Label1.Text = "Promedio: " & promedio.ToString("F2")
End Sub
```

Explicación:
Calcula el promedio de tres números ingresados.

---

32. Cambiar color de texto con ColorDialog

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Using cd As New ColorDialog()
        If cd.ShowDialog() = DialogResult.OK Then
            Label1.ForeColor = cd.Color
        End If
    End Using
End Sub
```

Explicación:
Permite al usuario seleccionar un color para el texto del Label.

---

33. Validar que TextBox no esté vacío antes de continuar

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    If String.IsNullOrWhiteSpace(TextBox1.Text) Then
        MessageBox.Show("Por favor ingrese un valor.")
    Else
        Label1.Text = "Entrada válida: " & TextBox1.Text
    End If
End Sub
```

Explicación:
Verifica que el usuario haya escrito algo antes de continuar.

---

34. Añadir elementos de array a ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim frutas() As String = {"Manzana", "Banana", "Naranja", "Pera"}

    ListBox1.Items.AddRange(frutas)

End Sub
```

Explicación:
Agrega varios elementos de un array al ListBox.

---

35. Seleccionar todo el texto de un TextBox con botón

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    TextBox1.SelectAll()

    TextBox1.Focus()

End Sub
```

Explicación:
Selecciona todo el texto dentro del TextBox para facilitar copiar o modificar.

---

36. Convertir texto a minúsculas

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Text = TextBox1.Text.ToLower()

End Sub
```

Explicación:
Convierte el texto a minúsculas y lo muestra.

---

37. Mostrar posición del cursor en TextBox

```
Private Sub TextBox1_KeyUp(sender As Object, e As KeyEventArgs) Handles
TextBox1.KeyUp

    Label1.Text = "Posición cursor: " & TextBox1.SelectionStart

End Sub
```

Explicación:
Muestra en tiempo real la posición del cursor en el TextBox.

---

38. Contar elementos en ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Label1.Text = "Elementos en lista: " & ListBox1.Items.Count

End Sub
```

Explicación:
Muestra cuántos elementos tiene el ListBox.

---

39. Eliminar elemento seleccionado en ListBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    If ListBox1.SelectedIndex <> -1 Then

        ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)

    Else

        MessageBox.Show("Seleccione un elemento para eliminar.")

    End If

End Sub
```

Explicación:
Elimina el elemento seleccionado del ListBox.

---

40. Mostrar texto modificado en RichTextBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    RichTextBox1.Text = TextBox1.Text.Replace("a", "@")

End Sub
```

Explicación:
Reemplaza todas las 'a' por '@' y muestra el resultado en un RichTextBox.

Programas de dificultad avanzado (41–60)

---

41. Temporizador que actualiza Label cada segundo

```
Private WithEvents timer As New Timer With {.Interval = 1000}


Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    timer.Start()
End Sub


Private Sub timer_Tick(sender As Object, e As EventArgs) Handles timer.Tick
    Label1.Text = DateTime.Now.ToString("HH:mm:ss")
End Sub
```

Explicación:
Un Timer actualiza el Label cada segundo con la hora actual.

---

42. Validar correo electrónico con expresión regular

```
Imports System.Text.RegularExpressions


Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim email As String = TextBox1.Text
    Dim pattern As String = "^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$"
    If Regex.IsMatch(email, pattern) Then
        Label1.Text = "Correo válido"
    Else
        Label1.Text = "Correo inválido"
    End If
End Sub
```

Explicación:
Usa regex para validar el formato del correo ingresado.

---

43. Arrastrar y soltar texto entre TextBoxes

```
Private Sub TextBox1_MouseDown(sender As Object, e As MouseEventArgs) Handles
TextBox1.MouseDown
    TextBox1.DoDragDrop(TextBox1.Text, DragDropEffects.Copy)
End Sub
```

```
Private Sub TextBox2_DragEnter(sender As Object, e As DragEventArgs) Handles
TextBox2.DragEnter
    If e.Data.GetDataPresent(DataFormats.Text) Then
        e.Effect = DragDropEffects.Copy
    End If
End Sub


Private Sub TextBox2_DragDrop(sender As Object, e As DragEventArgs) Handles
TextBox2.DragDrop
    TextBox2.Text = e.Data.GetData(DataFormats.Text).ToString()
End Sub
```

Explicación:
Permite arrastrar texto de un TextBox y soltarlo en otro.

---

44. Graficar valores en un Chart

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Chart1.Series.Clear()
    Chart1.Series.Add("Ventas")
    Chart1.Series("Ventas").ChartType =
DataVisualization.Charting.SeriesChartType.Column
    Chart1.Series("Ventas").Points.AddXY("Enero", 100)
    Chart1.Series("Ventas").Points.AddXY("Febrero", 150)
    Chart1.Series("Ventas").Points.AddXY("Marzo", 130)
End Sub
```

Explicación:
Muestra un gráfico de columnas con datos predefinidos.

---

45. Crear formulario modal y mostrar datos

```
' Form2.vb
Public Class Form2
    Public Property Mensaje As String
    Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Label1.Text = Mensaje
    End Sub
End Class
```

```
' Form1.vb

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim f2 As New Form2()

    f2.Mensaje = "Hola desde Form1"

    f2.ShowDialog()

End Sub
```

Explicación:
Abre un formulario modal y pasa datos a él.

---

46. Guardar texto de RichTextBox a archivo

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Using sfd As New SaveFileDialog()

        sfd.Filter = "Archivos de texto|*.txt"

        If sfd.ShowDialog() = DialogResult.OK Then

            System.IO.File.WriteAllText(sfd.FileName, RichTextBox1.Text)

        End If

    End Using

End Sub
```

Explicación:
Permite guardar el contenido del RichTextBox en un archivo.

---

47. Leer archivo de texto y mostrar en RichTextBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Using ofd As New OpenFileDialog()

        ofd.Filter = "Archivos de texto|*.txt"

        If ofd.ShowDialog() = DialogResult.OK Then

            RichTextBox1.Text = System.IO.File.ReadAllText(ofd.FileName)

        End If

    End Using

End Sub
```

Explicación:
Carga el contenido de un archivo de texto en un RichTextBox.

---

48. Validar contraseña y mostrar mensaje

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim pwd As String = TextBox1.Text

    If pwd.Length >= 8 AndAlso pwd.Any(Function(c) Char.IsDigit(c)) AndAlso
pwd.Any(Function(c) Char.IsUpper(c)) Then

        Label1.Text = "Contraseña válida"

    Else

        Label1.Text = "Contraseña debe tener al menos 8 caracteres, un dígito y
una mayúscula"

    End If

End Sub
```

Explicación:
Valida que la contraseña cumpla con ciertos requisitos.

---

49. Mostrar progreso con ProgressBar

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    ProgressBar1.Value = 0

    For i As Integer = 1 To 100

        ProgressBar1.Value = i

        Application.DoEvents() ' Actualiza la interfaz

        Threading.Thread.Sleep(10)

    Next

End Sub
```

Explicación:
Simula una tarea mostrando progreso en una barra.

---

50. Buscar texto en RichTextBox y resaltarlo

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

    Dim texto As String = TextBox1.Text

    Dim startIndex As Integer = 0

    RichTextBox1.SelectAll()

    RichTextBox1.SelectionBackColor = Color.White


    While startIndex < RichTextBox1.TextLength
```

```
        Dim index As Integer = RichTextBox1.Find(texto, startIndex,
RichTextBoxFinds.None)
        If index = -1 Then Exit While
        RichTextBox1.Select(index, texto.Length)
        RichTextBox1.SelectionBackColor = Color.Yellow
        startIndex = index + texto.Length
    End While
End Sub
```

Explicación:
Busca y resalta todas las ocurrencias de un texto en el RichTextBox.

---

51. Cambiar tamaño de ventana con TrackBar

```
Private Sub TrackBar1_Scroll(sender As Object, e As EventArgs) Handles
TrackBar1.Scroll
    Me.Width = TrackBar1.Value * 10
End Sub
```

Explicación:
Modifica el ancho del formulario según la posición del TrackBar.

---

52. Mostrar ToolTip en botón

```
Private toolTip1 As New ToolTip()

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    toolTip1.SetToolTip(Button1, "Este es un botón para ejecutar una acción")
End Sub
```

Explicación:
Muestra un tooltip explicativo al pasar el mouse sobre el botón.

---

53. Ordenar ListView por columna

```
Private Sub ListView1_ColumnClick(sender As Object, e As ColumnClickEventArgs)
Handles ListView1.ColumnClick
    ListView1.ListViewItemSorter = New ListViewItemComparer(e.Column)
End Sub

Class ListViewItemComparer
    Implements IComparer
```

```
    Private col As Integer
    Public Sub New(column As Integer)
        col = column
    End Sub

    Public Function Compare(x As Object, y As Object) As Integer Implements
IComparer.Compare
        Return [String].Compare(CType(x, ListViewItem).SubItems(col).Text,
CType(y, ListViewItem).SubItems(col).Text)
    End Function
End Class
```

Explicación:
Permite ordenar los elementos del ListView al hacer clic en la cabecera.

---

54. Bloquear TextBox tras 3 intentos fallidos

```
Dim intentos As Integer = 0

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    If TextBox1.Text = "clave123" Then
        MessageBox.Show("Acceso concedido")
        intentos = 0
    Else
        intentos += 1
        MessageBox.Show("Clave incorrecta")
        If intentos >= 3 Then
            TextBox1.Enabled = False
            Button1.Enabled = False
            MessageBox.Show("Has excedido el número de intentos")
        End If
    End If
End Sub
```

Explicación:
Deshabilita el TextBox y botón tras 3 intentos erróneos.

---

55. Mostrar imagen desde recursos y cambiar con ComboBox

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBox1.SelectedIndexChanged
```

```
    Select Case ComboBox1.SelectedItem.ToString()
        Case "Gato"
            PictureBox1.Image = My.Resources.gato
        Case "Perro"
            PictureBox1.Image = My.Resources.perro
        Case Else
            PictureBox1.Image = Nothing
    End Select
End Sub
```

Explicación:
Muestra una imagen dependiendo de la selección en el ComboBox.

---

56. Manejar evento DragDrop para archivos en formulario

```
Private Sub Form1_DragEnter(sender As Object, e As DragEventArgs) Handles
Me.DragEnter
    If e.Data.GetDataPresent(DataFormats.FileDrop) Then
        e.Effect = DragDropEffects.Copy
    End If
End Sub


Private Sub Form1_DragDrop(sender As Object, e As DragEventArgs) Handles
Me.DragDrop
    Dim archivos() As String = e.Data.GetData(DataFormats.FileDrop)
    MessageBox.Show("Archivo arrastrado: " & archivos(0))
End Sub
```

Explicación:
Permite arrastrar un archivo al formulario y muestra su nombre.

---

57. Mostrar diálogo de impresión

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Using pd As New PrintDialog()
        If pd.ShowDialog() = DialogResult.OK Then
            ' Aquí iría la impresión real
            MessageBox.Show("Imprimiendo...")
        End If
    End Using
```

```
End Sub
```

Explicación:
Abre el diálogo estándar de impresión (sin implementar la impresión real).

---

58. Crear menú contextual para TextBox

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim menu As New ContextMenuStrip()
    menu.Items.Add("Copiar", Nothing, AddressOf CopiarTexto)
    menu.Items.Add("Pegar", Nothing, AddressOf PegarTexto)
    TextBox1.ContextMenuStrip = menu
End Sub

Private Sub CopiarTexto(sender As Object, e As EventArgs)
    Clipboard.SetText(TextBox1.SelectedText)
End Sub

Private Sub PegarTexto(sender As Object, e As EventArgs)
    TextBox1.Paste()
End Sub
```

Explicación:
Agrega un menú contextual con opciones Copiar y Pegar al TextBox.

---

59. Mostrar formulario hijo dentro de formulario padre (MDI)

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim child As New Form()
    child.MdiParent = Me
    child.Text = "Formulario hijo"
    child.Show()
End Sub
```

Explicación:
Crea y muestra un formulario hijo dentro de un contenedor MDI.

---

60. Guardar configuración usando My.Settings

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    My.Settings.Usuario = TextBox1.Text
```

```
    My.Settings.Save()
    MessageBox.Show("Configuración guardada")
End Sub

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    TextBox1.Text = My.Settings.Usuario
End Sub
```

Explicación:
Guarda y carga una configuración simple usando My.Settings.

¿Quieres que te prepare todos estos programas en un archivo para que lo tengas cómodo o que siga ayudándote con más ejemplos?

34. Controles Visual Basic net

🟢 Controles Básicos

Control	Descripción
`Label`	Muestra texto estático. Ideal para títulos o instrucciones.
`TextBox`	Permite al usuario introducir texto.
`Button`	Ejecuta una acción al hacer clic.
`CheckBox`	Opción con casilla para marcar o desmarcar.
`RadioButton`	Botón de selección exclusiva dentro de un grupo.
`PictureBox`	Muestra imágenes (BMP, JPG, PNG, etc.).
`GroupBox`	Agrupa controles visuales relacionados.
`Panel`	Contenedor para organizar otros controles.
`ListBox`	Lista de elementos con selección.
`ComboBox`	Lista desplegable de opciones.

🟡 Controles Intermedios

Control	Descripción
`DateTimePicker`	Permite seleccionar una fecha (y hora si se configura).
`MaskedTextBox`	TextBox con máscara de entrada (ej: fechas, DNI, etc.).
`TrackBar`	Selector de valores con una barra deslizante.
`ProgressBar`	Barra de progreso visual para tareas.
`Timer`	Ejecuta tareas periódicas en intervalos definidos.
`TabControl`	Permite crear pestañas con diferentes contenidos.
`ListView`	Lista avanzada con columnas, imágenes y más formatos.
`TreeView`	Muestra estructura jerárquica (como carpetas).
`RichTextBox`	Similar al TextBox, pero permite formato de texto enriquecido.
`ToolTip`	Muestra ayuda emergente al pasar el mouse por encima de controles.

🔴 Controles Avanzados

Control	Descripción
`DataGridView`	Tabla de datos editable. Ideal para mostrar información estructurada.
`OpenFileDialog`	Diálogo para abrir archivos.
`SaveFileDialog`	Diálogo para guardar archivos.
`ColorDialog`	Permite elegir un color desde una paleta.
`FontDialog`	Permite seleccionar tipografía.
`MenuStrip`	Barra de menús (Archivo, Editar, etc.).
`ContextMenuStrip`	Menú contextual (clic derecho) para cualquier control.
`StatusStrip`	Muestra información de estado en la parte inferior del formulario.
`NotifyIcon`	Icono en la bandeja del sistema con notificaciones.
`Chart`	Control para representar gráficos (barras, líneas, pastel, etc.).

¿Qué más necesitas?

Puedo ayudarte con:

- Crear ejemplos para cualquiera de estos controles.
- Explicar eventos y propiedades específicas.
- Armar una guía en PDF con prácticas para aprender a usarlos.
- Diseñar una mini aplicación usando varios controles combinados.

Apendice

En **Visual Basic .NET**, si estás trabajando con **una aplicación de consola**, puedes **limpiar la pantalla** usando:

```
Console.Clear()
```

Ejemplo:

```
Module Module1
    Sub Main()
        Console.WriteLine("Texto inicial...")
        Console.ReadKey() ' Espera una tecla
        Console.Clear()   ' Limpia la pantalla
        Console.WriteLine("Pantalla limpia.")
        Console.ReadKey()
    End Sub
End Module
```

Si en cambio estás trabajando con una **aplicación Windows Forms**, "limpiar la pantalla" depende de lo que quieras borrar:

En Windows Forms:

- Para **limpiar un `TextBox`**:

  ```
  TextBox1.Clear()
  ```

- Para **limpiar un `Label`**:

  ```
  Label1.Text = ""
  ```

- Para **limpiar varios controles de un formulario**, puedes recorrerlos:

  ```
  For Each ctrl As Control In Me.Controls
      If TypeOf ctrl Is TextBox Then
          CType(ctrl, TextBox).Clear()
      ElseIf TypeOf ctrl Is Label Then
          CType(ctrl, Label).Text = ""
      End If
  Next
  ```

Ejemplo básico: Ejecutar un `.exe` externo

```
Imports System.Diagnostics

Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        Try
            Process.Start("C:\Ruta\al\archivo\programa.exe")
        Catch ex As Exception
            MessageBox.Show("Error al ejecutar el programa: " & ex.Message)
        End Try
    End Sub
End Class
```

Detalles:

- Reemplaza `"C:\Ruta\al\archivo\programa.exe"` por la ruta real al ejecutable que quieres lanzar.
- Puedes usar una **ruta relativa** si el `.exe` está en el mismo directorio que tu aplicación.
- Este código se ejecuta, por ejemplo, al hacer clic en un botón (`Button1`).

Ejemplo avanzado: pasar argumentos al .exe

```
Dim miProceso As New ProcessStartInfo()
miProceso.FileName = "C:\Ruta\programa.exe"
miProceso.Arguments = "-a -b valor"
Process.Start(miProceso)
```

Recomendaciones:

- Asegúrate de que el `.exe` que vas a ejecutar no requiera privilegios de administrador, o tendrás que ejecutarlo con derechos elevados.
- Si necesitas **esperar a que termine**, puedes usar:

```
Dim proceso As Process = Process.Start("C:\Ruta\programa.exe")
proceso.WaitForExit()
```

En Visual Basic .NET (VB.NET), para interactuar entre formularios (Forms) como abrir uno nuevo y cerrar el actual, se utilizan los métodos `Show()`, `ShowDialog()`, y `Close()`.

---

1. Abrir un nuevo formulario desde otro

Supongamos que tienes dos formularios: `Form1` y `Form2`.

Desde `Form1`, puedes abrir `Form2` así:

```
' Abre Form2 y deja Form1 abierto
Dim f2 As New Form2()
f2.Show()
' Abre Form2 y bloquea Form1 hasta que cierres Form2
Dim f2 As New Form2()
f2.ShowDialog()
```

---

2. Cerrar el formulario actual

Para cerrar un formulario:

```
Me.Close()
```

Esto cerrará el formulario en el que está el código (`Me` hace referencia al formulario actual).

---

3. Abrir un formulario y cerrar el actual al mismo tiempo

Desde `Form1`, si quieres abrir `Form2` y cerrar `Form1`, puedes hacer esto:

```
Dim f2 As New Form2()
f2.Show()
Me.Close()
```

> ⚠ **Nota importante**: Si `Form1` es el formulario de inicio del proyecto (Startup Form), al cerrarlo se cerrará la aplicación entera **a menos que configures lo siguiente:**

---

4. Cambiar el comportamiento del cierre del formulario principal

Para que **al cerrar un formulario no se cierre toda la aplicación**, haz esto:

En `My Project > Application`:

- Cambia la opción **"Cerrar aplicación cuando se cierre el formulario de inicio"** a **desmarcada**.

---

5. Alternar entre formularios desde botones

En `Form1`:

```
Private Sub btnAbrirForm2_Click(sender As Object, e As EventArgs) Handles
btnAbrirForm2.Click

    Dim f2 As New Form2()

    f2.Show()

    Me.Hide() ' O usar Me.Close() si quieres cerrar completamente

End Sub
```

En `Form2`:

```
Private Sub btnVolverForm1_Click(sender As Object, e As EventArgs) Handles
btnVolverForm1.Click

    Dim f1 As New Form1()

    f1.Show()

    Me.Close()

End Sub
```

---

ejemplo completo paso a paso en Windows Forms con Visual Basic .NET, donde puedes:

- Abrir `Form2` desde `Form1`
- Volver de `Form2` a `Form1`
- Cerrar y abrir formularios correctamente sin cerrar toda la aplicación.

Paso 1: Crear el proyecto

1. Abre **Visual Studio**.
2. Crea un nuevo proyecto de tipo **"Aplicación de Windows Forms (.NET Framework)"** en **Visual Basic**.
3. Nómbralo por ejemplo: `FormularioEjemplo`.

Paso 2: Agrega los formularios

- Por defecto ya tendrás un formulario llamado `Form1.vb`.
- Agrega un nuevo formulario:
    - Haz clic derecho en el proyecto > **Agregar > Windows Form**.
    - Llama al nuevo formulario `Form2.vb`.

Paso 3: Diseña `Form1`

1. En `Form1`, agrega un botón desde la caja de herramientas.
2. Cambia el texto del botón a `Abrir Form2`.

Código de `Form1.vb`

```
Public Class Form1

    Private Sub btnAbrirForm2_Click(sender As Object, e As EventArgs) Handles
btnAbrirForm2.Click

        Dim f2 As New Form2()

        f2.Show()

        Me.Hide()

    End Sub

End Class
```

Asegúrate de que el botón se llame `btnAbrirForm2`.

Paso 4: Diseña `Form2`

1. En `Form2`, agrega un botón.
2. Cámbiale el texto a `Volver a Form1`.

Código de `Form2.vb`

```
Public Class Form2

    Private Sub btnVolverForm1_Click(sender As Object, e As EventArgs) Handles
btnVolverForm1.Click

        Dim f1 As New Form1()

        f1.Show()

        Me.Close()

    End Sub

End Class
```

Asegúrate de que el botón se llame `btnVolverForm1`.

Paso 5: Configura para que no se cierre toda la aplicación al cerrar un form

1. Ve al menú **Proyecto > Propiedades de [TuProyecto]**.
2. En la pestaña **Aplicación**:
    - Desmarca la opción **"Cerrar aplicación cuando se cierre el formulario de inicio"**.
    - Asegúrate de que el formulario de inicio sea `Form1`.

Resultado

- Al ejecutar, `Form1` se abre.
- Al pulsar el botón, se abre `Form2` y se oculta `Form1`.
- Al volver desde `Form2`, se abre `Form1` nuevamente y se cierra `Form2`.

Epílogo

Hemos llegado al final de este recorrido por el lenguaje Visual Basic .NET. A lo largo de estas páginas hemos explorado conceptos fundamentales, estructuras lógicas, controles gráficos, técnicas de programación orientada a objetos y mucho más. Pero más allá del contenido técnico, mi deseo ha sido transmitir una idea más profunda: que programar no es solo escribir líneas de código, sino una forma de pensar, de crear y de resolver problemas.

Visual Basic .NET, a pesar del paso del tiempo y de la aparición de nuevos lenguajes, sigue siendo una herramienta poderosa y accesible. Su claridad y su capacidad para desarrollar aplicaciones de escritorio siguen vigentes, y para muchos —como fue mi caso— representa una puerta de entrada maravillosa al mundo de la programación.

Este libro es parte de un proyecto más amplio: una serie de textos que escribo con el corazón, con la intención de compartir lo que he aprendido. Porque sigo creyendo firmemente que el conocimiento no es algo que se posee, sino algo que se transmite. Y solo cuando lo compartimos, el conocimiento adquiere su verdadero valor: el de hacernos libres.

Si llegaste hasta aquí, gracias. Gracias por confiar en este libro y en mi experiencia. Espero que el aprendizaje te haya sido útil y que te sirva como base para seguir creciendo como programador o programadora. Si alguna de estas páginas logró despertar en ti la misma pasión que yo sentí con mis primeros pasos en QBasic, entonces este esfuerzo ya ha valido la pena.

Recuerda: sigue aprendiendo, sigue explorando, y no tengas miedo de equivocarte. Cada error es una oportunidad para entender mejor, para mejorar, y para crecer.

Y como al principio, termino también con un agradecimiento: a mi familia, que me acompaña siempre, y a mi querida Marisa, cuya presencia ha sido luz y apoyo constante en cada etapa de este camino.

Hasta el próximo libro.